

Simultaneous Localization and Planning for Physical Mobile Robots via Enabling Dynamic Replanning in Belief Space

Ali-akbar Agha-mohammadi, Saurav Agarwal, Suman Chakravorty, and Nancy M. Amato

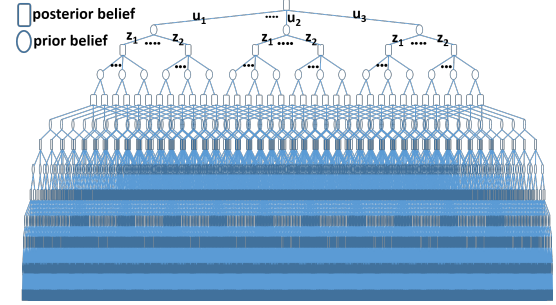
Abstract—Simultaneous planning while localizing is a crucial ability for an autonomous robot operating under uncertainty. This paper addresses this problem by designing methods to dynamically replan while the localization uncertainty or environment map is updated. In particular, relying on sampling-based methods, the proposed online planning scheme can cope with challenging situations, including when the localization update or changes in the environment alter the homotopy class of trajectories, in which the optimal plan resides. The proposed algorithm significantly reduces the need for stabilization in the state-of-the-art FIRM (Feedback-based Information RoadMap) method. As a result, it outperforms FIRM’s performance and success probability. Applying belief space planning to physical systems brings with it a plethora of challenges. Thus, designing computationally tractable algorithms, a key focus and contribution of this paper is to implement the proposed planner on a physical robot and show the SLAP (simultaneous localization and planning) performance under uncertainty, in changing environments, and in the presence of large disturbances such as kidnapped robot situation.

Index Terms—Motion planning, belief space, robust, POMDP, uncertainty, mobile robots, rollout

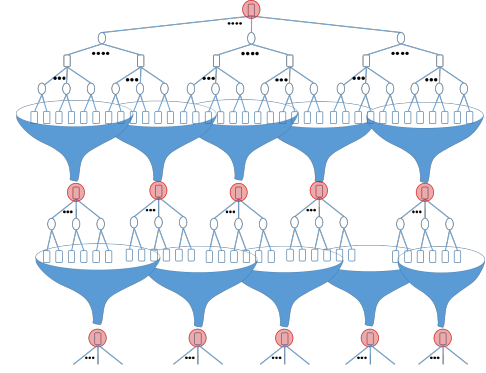
I. INTRODUCTION

Simultaneous Localization and Planning (SLAP) refers to the ability of autonomous robots to (re)plan dynamically every time the localization module updates the probability distribution on the robot’s state. For autonomous robots to reliably operate in a real-world environments, online (re)planning under uncertainty is a key requisite to enable SLAP and cope with uncertainties and changes in the environment. For example, consider a low-cost mobile robot, working in an office environment, where its goals (tasks) are assigned online based on user requests, while coping with changes in the obstacle map (e.g., office doors switch state between open and closed). Such changes in the obstacle map or in the goal location often call for global replanning as they switch the optimal homotopy class of solutions from one to another. Therefore, the robot needs to change its motion plan in real-time as it encounters changes in its goal or in the environment. What makes such online replanning challenging is that low-cost robots are not able to follow the control commands exactly due to motion noise and they do not have perfect measurements due to sensor noise. Therefore, this problem calls for online planning algorithms in uncertain, partially observable environments. In a broader sense, this problem is an instance of the problem of decision making and control under uncertainty.

In general, decision making and control under uncertainty is a ubiquitous challenge in many robotic applications. For an autonomous robot to operate reliably, it is crucial to be able to perceive sensory measurements, infer its situation (state) in the environment, and plan and take actions accordingly. However, in



(a) Belief tree: forward construction



(b) Belief graph: backward construction

Fig. 1. (a) This figure depicts a typical search tree in belief space, corresponding to a really small problem with 3 actions $\mathbb{U} = \{u_1, u_2, u_3\}$ and two observations $\mathbb{Z} = \{z_1, z_2\}$. Each posterior distribution/belief branches into $|\mathbb{U}|$ number of priors and each prior belief branches into $|\mathbb{Z}|$ posteriors, and thus the tree grows exponentially in the search depth. (b) This figure depicts the idea of using funnels (local feedback controllers) in belief space that can break this exponential growth by funneling a large set of posteriors into a pre-computed beliefs (in red circles). Thus a graph is formed in belief space with funnels as edges and the pre-computed beliefs as nodes. The graph grows linearly with the search depth.

partially-observable environments the state of the system cannot be determined exactly due to imperfect and noisy measurements. Based on the system’s model, a filtering module (e.g., Kalman filter) can provide an estimation of the state, i.e., a probability distribution function (pdf) over all possible system states. This pdf describing the localization uncertainty is also referred to as *belief* or *information-state*. Based on the belief at every time step actions are chosen. To formalize this problem of finding the optimal mapping between perceived observations and the taken action, we rely on the most general formulation, i.e., Partially-Observable Markov Decision Processes (POMDP) [34, 61].

There are a number of challenges in dealing with POMDPs, including the *curse of dimensionality* and *curse of history*. Curse of dimensionality refers to the high dimensions of the belief space. If the underlying robotic system evolves in a discrete grid world with n cells, the corresponding belief space is an n -dimensional continuous space. Moreover, if the underlying state space is continuous (which is the case for most real robotic

A. Agha is with Qualcomm Research, San Diego, CA, 92121. S. Agarwal and S. Chakravorty are with the Dept. of Aerospace Eng. and N. Amato is with the Dept. of Computer Science & Eng. at Texas A&M University, College Station, TX 77843, USA. Emails: aliagha@qualcomm.edu, {sauravag, schakrav, amato}@tamu.edu

applications), then the belief space is infinite dimensional. Methods such as [10, 30, 39, 41, 44, 50, 51, 62, 65] alleviate these issues and take POMDPs to more challenging and realistic problems. In this paper, we consider a class of POMDPs that commonly arise in modeling the SLAP problem. The settings are similar to the ones used in KF-based localization literature [26, 66], such as (i) the system model is given as closed-form nonlinear equations, (ii) the state/action/observation spaces are continuous, and (iii) belief is unimodal, thus it is well-approximated by a Gaussian.

In addition to above-mentioned challenges associated with POMDPs, when dealing with real-world physical systems, another important challenge is the discrepancy between the real models with the models used for computation, such as discrepancies in the environment map, the system and sensor models, and noise models. Such discrepancies can lead to deviations of the system from the desired plan. A plausible solution for this problem is an ability to carry out planning in a simultaneous manner with localization, i.e., an ability to replan dynamically to cope with changes in the environment and deviations resulted from model discrepancies and intermittent sensing failures.

To enable an online replanning scheme for SLAP, we rely on multi-query methods in belief space and specifically the Feedback-based Information Roadmap (FIRM) method, as discussed below. The main body of POMDP literature, in particular sampling-based methods, propose single-query solvers, i.e., the computed solution depends on the initial belief [41, 54, 68]. Therefore, in replanning (planning from a new initial belief) almost all the computations need to be reproduced, which limits their usage in solving SLAP where dynamic replanning is required. However, multi-query methods such as FIRM provide a construction mechanism, independent of the initial belief of the system (Fig. 1 and 3), making them suitable methods for SLAP. The original FIRM framework provided a reliable methodology for solving the problem of motion planning under uncertainty by reducing the intractable dynamic programming (DP) to a tractable DP over the nodes of the FIRM graph. In this paper, we extend our previous work on FIRM by proposing a dynamic replanning scheme in belief space that enables online replanning for real world applications in mobile robotics. This extension leads to intelligent behaviors that provably takes the solution closer to the optimal and can guarantee the success probability only increases via this extension. In addition to theoretical results on the online generalization of FIRM, the main emphasis of this paper is on the implementation of the proposed SLAP solution on a physical robot. We investigate how dynamic online replanning can generate a feedback plan that leads to higher performance and success probability. Also, we demonstrate its unique ability to cope with discrepancies between real models and computational models such as changes in the environment and large deviations which can globally change the plan by changing the optimal homotopy class of solutions. We believe these results lay the ground work for advancing the theoretical POMDP framework towards practical SLAP applications, and achieving long-term autonomy in robotic systems.

A. Related Work

Online replanning in belief space is a vital capability to solve the SLAP problem for two main reasons: First, belief dynamics are usually more random than state dynamics because the belief is directly affected by the measurement noise. Therefore,

a noisy measurement or spurious data association can cause large changes in the belief. Second, in practical applications, discrepancies between real and computation models are a significant source of error and cause the belief to occasionally have behaviors different than expected nominal behavior. Thus, simultaneously replanning as localizing, gives a system the ability to recover from such deviations. In addition, enabling SLAP can help the robot to cope with changes in the environment as well as recover from large unexpected deviations in the robot’s pose.

Active localization: Solving the planning problem alongside localization and mapping has been the topic of several recent works (e.g., [21], [20], [19], [36]). The method in [33] presents an approach to uncertainty-constrained simultaneous planning, localization and mapping for unknown environments and in [18], the authors propose an approach to actively explore unknown maps while imposing a hard constraint on the localization uncertainty at the end of the planning horizon. Our work assumes the environment map is known, formulates the problem in its most general form (POMDP), and focuses on online replanning in the presence of obstacles (possibly changing).

General-purpose POMDP solvers: There is a strong body of literature on general purpose POMDP solvers (e.g., [40], [9], [23], [63], [48]). We divide the literature on general purpose POMDPs to two main categories: The first category is offline solvers, whose goal is to compute a policy (e.g., [39, 51, 62, 65]). A survey paper on a class of these methods relying on point-based solvers is [59]. The second category is online search algorithms. In these methods instead of a policy (action for all possible beliefs), the goal is to find the best action for the current belief using a forward search tree rooted in the current belief. [56] surveys recent methods in this category. In recent years, general-purpose online solvers have become faster and more efficient. AEMS [55], DESPOT [64], ABT [38], and POMCP [60] are among the most successful methods in this category. Direct application of these methods to SLAP-like problems is a challenge due to (i) Expensive simulation steps, (ii) continuous, high-dimensional spaces, (iii) difficult tree pruning steps. We discuss these tree challenges in the following paragraphs.

Majority of above-mentioned methods rely on simulating the POMDP model forward in time and create a tree of possible scenarios in future. At each simulation step $(x', z, c) \sim \mathcal{G}(x, u)$, the simulator \mathcal{G} simulates taking action u at state x and computes the next state x' , observation z , and the cost and constraints of this transition. When dealing with Games (e.g., Go) or traditional POMDP problems (e.g., RockSample [56]), the forward simulation step and cost computation is computationally very inexpensive. However, in SLAP-like problems computing costs are typically much more expensive. An important example cost is a boolean value that determines if the system had collided with obstacles or not during this transition. This a very expensive computation in robotics applications, that restricts the application of tree-based methods to cases where computing collision probabilities is needed.

The second challenge in applying tree-based methods to SLAP is the “low chance of revisiting the same belief”. Tree-based methods require the simulator to revisit the same belief many times to learn its value. However, in SLAP-like problems with continuous state/action/observation spaces, the chances of visiting the same belief is almost zero.

Even the discretized version of the problem is really large. For a full tree of depth d , the number of simulation steps

along the tree is of the order $n_{cost} = O((|\mathcal{U}||\mathcal{Z}|)^d)$. Represented belief by a minimum n_b number of particles (to ensure accurate computation of collision probability), one needs to perform $n_{coll} = O(n_b(|\mathcal{U}||\mathcal{Z}|)^d)$ collision checks $((x', z, c) \sim \mathcal{G}(x, u))$ to construct the full tree. To provide some intuition on the size and type of the SLAP problem in this paper, where robot is working in an office-like environment localizing itself using visual landmarks, we report typical values in our experiments: We execute around 100 macro-actions, each of which has around 100 primitive actions. So, the scenario depth is in the order of $d = 10^4$ steps. Typical number of particles is in the order of $n_b = 100$. Our action, observation, and state spaces are continuous $|\mathcal{X}| = |\mathcal{U}| = |\mathcal{Z}| = \infty$ but a reasonable discretization could be of the order of $|\mathcal{U}| = 50^2$ where the speed of each wheel is discretized to 50 region. Assuming the robot can see 5 landmarks at any given time and each landmark observation is 2 dimensional (range and bearing), where each dimension is discretized to 100 steps, we have $|\mathcal{Z}| = 100^{10}$. Thus, the chances of revising the same belief in the discretized version of the problem is really low.

Finally, unlike many traditional POMDP domains where the domain structure (e.g., game rules) prunes a lot of tree branches, pruning the search tree is much more challenging in SLAP-like problems. For example consider motion planning problem where there exists a grid of homotopy classes amid obstacles. So, these are exponential number of paths from one corner to the opposite corner of the grid. Now, imagine that at the end of this grid, there is a narrow passage that all these paths has to go through to reach the goal. Thus, none of these paths (tree branches) can be pruned, because to compute which path is better, the belief has to be propagated over all of these paths to see which one leads to a smaller collision probability at the final narrow passage. Note that the problem has a terminal belief and there is no discount factor.

Thus, clearly applying exact general-purpose POMDP solvers to this type of problem is a challenge, mainly because one needs to compute accurate costs (e.g., collision probabilities) very deep in the search tree with high accuracy. To be able to tackle this problem, we exploit the additional structure existing in the SLAP problem, including knowledge of closed form dynamics, and sensor model to design closed loop controllers. We further restrict the scope of the problem to Gaussian belief space. Finally, we add additional structure (funnels) to the problem that leads to suboptimal solutions but allows us to solve the problem and provide guarantees on the safety constraints (collision probabilities). Further, using rollout methods, we take this suboptimal solution closer to the optimal solution by bypassing unnecessary of funnels in the online phase. It is worth noting that most of the above-mentioned tree-based methods are complementary to the proposed method and can be combined with the ideas presented in this paper. We discuss some potentially useful combinations in the future work section.

Continuous Gaussian POMDP solvers: A different class of methods restrict the form of uncertainty to Gaussian and extend the traditional deterministic motion planning methods (e.g., PRM and RRT) to belief space. Examples include [54] [68], [16]. More recently, [69] proposes an efficient approximate value iteration method. Starting from an initial solution (trajectory in belief space), it converges to the closest local minimum to the initial solution. These methods are single query (the solution is valid for a given initial belief), thus in case of replanning

from a new belief most of the computation needs to be redone. Replanning becomes more challenging when the planner has to switch the plan from one homotopy class to another.

RHC with direct optimization: Another class of planners rely on the direct optimization methods in the Receding Horizon Control (RHC) manner. The optimization variable is typically an open-loop sequence of actions. The replanning algorithm in RHC can be recapped as: At every step a sequence of optimal actions is computed within a limited horizon of T steps. Then, only the first action is executed and the rest is discarded. The executed action takes the system to a new point and from this new point, a new sequence of optimal actions is recomputed within horizon T and this process is repeated until the system reaches the goal region. The RHC framework was originally designed for deterministic systems and its extension to stochastic systems and belief space planning is still an open problem. A direct approach is to replace the uncertain quantities (such as future observations) with their nominal values (e.g., most likely observations), and then treat the stochastic system as a deterministic one and use it in an RHC framework (e.g., [27], [53], [22], [31], [67]). However, in such an approach the optimization is carried out only within a limited horizon, and therefore the system may locally choose good actions but after a while may find itself in a high-cost region.

POMDP applied to physical robots: From an experimental point of view, a few recent work have focused on applying belief space planning to real-world robots. [7] implements a belief planner on a mobile manipulator with time of traversal as a cost metric. [35] is an integrated task and motion planner, utilizing symbolic abstraction, whose performance is demonstrated on a PR2 robot tasked with picking and placing household objects. In [14], the authors develop a stochastic motion planner and show its performance on a physical manipulation task where unknown obstacles are placed in the robot's operating space and the task-relevant objects are disturbed by external agents. [10] extends the application of POMDP methods to autonomous driving in a crowd, where an autonomous golf cart drives amid pedestrian and avoids them by predicting their intentions. Authors in [45] apply a POMDP-based planner to navigate a PR2 robot in an office-like environment. The paper proposes an elegant way of incorporating environment changes into the planning framework and can cope with changes in the homotopy class. The main difference with our method is that the authors are concerned about the uncertainty in obstacles rather than the robot and assume that the robot position in the map is perfectly known.

B. Highlights and Contributions

This paper extends our previous work in [2]. Compared to [2], we discuss in detail the concept of rollout-based belief space planning, policy execution, and present extensive simulation and experimental results to demonstrate the performance improvements made by using the proposed method. We also present analyses that guarantees a lower execution cost and failure probability as compared to the nominal FIRM policy. The main contributions, highlights, and impact of this work can be summarized as follows.

Online belief planner to enable SLAP: We propose an online planning method in belief space. The method lends itself to the class of rollout-based methods [11] and extends them to the belief space. Compared to belief space RHC methods, this

method is not limited to a horizon, does not get stuck in local minima, and does not assume deterministic future observations.

Online switching between homotopy classes: In motion planning, homotopy classes of paths refer to sets of paths that can be deformed into each other by continuous transformation (bending and stretching) without passing through obstacles [12]. A unique feature of the presented method is that it is capable of updating the plan globally online, even when the homotopy class of optimal solution has changed. This feature allows the proposed planner to work in changing environments and cope with large deviations.

Smart stabilization policy: The proposed method supercedes a state-of-the-art method, FIRM [6], in performance, success probability, and ability to cope with changing environments. It builds upon a FIRM and inherits the desired features of the FIRM framework such as robustness, scalability, and the feedback nature of the solution. But, it also significantly reduces the need for belief node stabilization in the original FIRM method to cases where it is absolutely necessary. Thus the proposed method can be viewed as a FIRM with smart selective stabilization policy. In the original FIRM framework, at every node the system needs to steer its sensory information to reach the belief node (each graph node is a belief, i.e., a particular localization uncertainty). But, in this paper, by embedding an online local planning module in the FIRM framework, we achieve a locally optimal tradeoff between stabilization to a node (i.e., exploring the information space to reach the exact belief node) and moving forward towards goal (exploiting the gradient of local cost function), while the global optimality on the graph is still guaranteed by solving dynamic programming. As a result of this optimal tradeoff, interesting behaviors emerge out of the algorithm without encoding any heuristic. These behaviors exchange information and energy. For example, consider a case when the desired cost is to “reach a goal while minimizing the probability of colliding with obstacles”. In that case, in the open areas where there are no narrow passages, the system bypasses the belief node stabilizations. It speeds up and does not waste any time gathering information and reducing its uncertainty as there is not much benefit in doing so in obstacle-free regions. However, once it faces with obstacles or narrow enough passages, it automatically decides to perform stabilization (partially) until the uncertainty is shrunk enough to safely traverse the narrow passage. Fig. 2, shows this phenomenon pictorially.

Performance guarantees: We provide lower bounds on the performance of the method and show that in stationary environments, the performance and success probability of the proposed method always exceeds (or in the worst case is equivalent to) those of the FIRM method.

Applications to physical systems: Among the set of methods that cope with continuous state/action/observation POMDP, only a very small number of methods (e.g., [7],[35], [10],[45]) have been applied to physical systems due to the computational complexity of these methods in real-world robotics problem. An important contribution of this work is to implement a continuous state/action/observation POMDP solver on a physical robot in a real-world office-like environment. We explain this procedure in details and discuss the theoretical tools and methods designed during the process of to help with the real-world implementing including the lazy feedback approach.

II. BELIEF SPACE PLANNING FOR MOBILE ROBOTS

In this section, we briefly describe the abstract framework of Feedback-based Information RoadMap (FIRM) followed by a description of its concrete implementation in our experiments. We refer the reader to [3, 6] for a more in-depth description of the abstract FIRM method.

A. Planning Under Uncertainty

In motion planning under uncertainty, we are looking for a policy π_k at time-step k that generates a control action u_k based on available information about the robot. We start by defining some key terms. Consider a system whose state is denoted by x_k at the k -th time step, let u_k and w_k , be the control action and motion noise respectively at time k . Let us denote the state evolution model by $x_{k+1} = f(x_k, u_k, w_k)$. In a partially observable system, we do not have access to the true state of the system. Rather, we get some noisy observations of the robot state. Let us denote the sensor measurement (or observation) vector by z_k at the k -th time step and the measurement model by $z_k = h(x_k, v_k)$, where v_k denotes sensing noise. The only data that is available for decision making at the k -th time step (i.e., generating u_k) is the history of observations and controls: $\mathcal{H}_k = \{z_{0:k}, u_{0:k-1}\} = \{z_0, z_1, \dots, z_k, u_0, \dots, u_{k-1}\}$.

Conditional probability distribution over all possible system states $b_k = p(x_k | \mathcal{H}_k)$, which is referred to as *belief* or *information-state* compresses the data \mathcal{H}_k . It is well-known that in Bayesian filtering, belief can be computed recursively based on the last action and current observation $b_{k+1} = \tau(b_k, u_k, z_{k+1})$ [11],[66]:

$$b_{k+1} = \alpha p(z_{k+1} | x_{k+1}) \int_{\mathbb{X}} p(x_{k+1} | x_k, u_k) b_k dx_k =: \tau(b_k, u_k, z_{k+1}).$$

where, $\alpha = p(z_{k+1} | \mathcal{H}_k, u_k)^{-1}$ is the normalization constant. As a result of filtering, the action u_k can be taken based on the belief b_k using a policy (planner) π_k , i.e., $u_k = \pi_k(b_k)$. It is well-known that π_k is the solution of a POMDP, which is intractable over continuous state/action/observation spaces.

B. A Brief Overview of FIRM

FIRM is a framework designed to reduce the mentioned intractable POMDP problem to a tractable problem, by generating a representative graph (PRM: Probabilistic Roadmap Method) in the belief space. Similar to PRMs where the *solution path* is a concatenation of local paths, in FIRM the *solution policy* is a concatenation of local policies. Every node in an FIRM is a small region $B = \{b : \|b - \hat{b}\| \leq \varepsilon\}$ around a sampled belief \hat{b} . We denote the i -th node by B^i and the set of nodes by $\mathbb{V} = \{B^i\}$. Each edge in an FIRM is a closed-loop local controller whose goal is to steer the belief into the target node of the edge. An edge controller connecting nodes i and j is denoted by μ^{ij} and the set of edges by $\mathbb{M} = \{\mu^{ij}\}$. A metaphor for each local controller is a “funnel in belief space”. As depicted in Fig. 3, each funnel steers the set of beliefs to a milestone belief. Further, using the slide-funnel composition, we can create sparse graphs of funnels as shown in Fig. 3. A basic instantiation of a funnel in belief space is stationary Linear Quadratic Gaussian (SLQG) controller (see Appendix C in [6]) and a basic instantiation of a slide in belief space is Time-Varying Linear Quadratic Gaussian (TV-LQG) controller (see Appendix B in [6]).

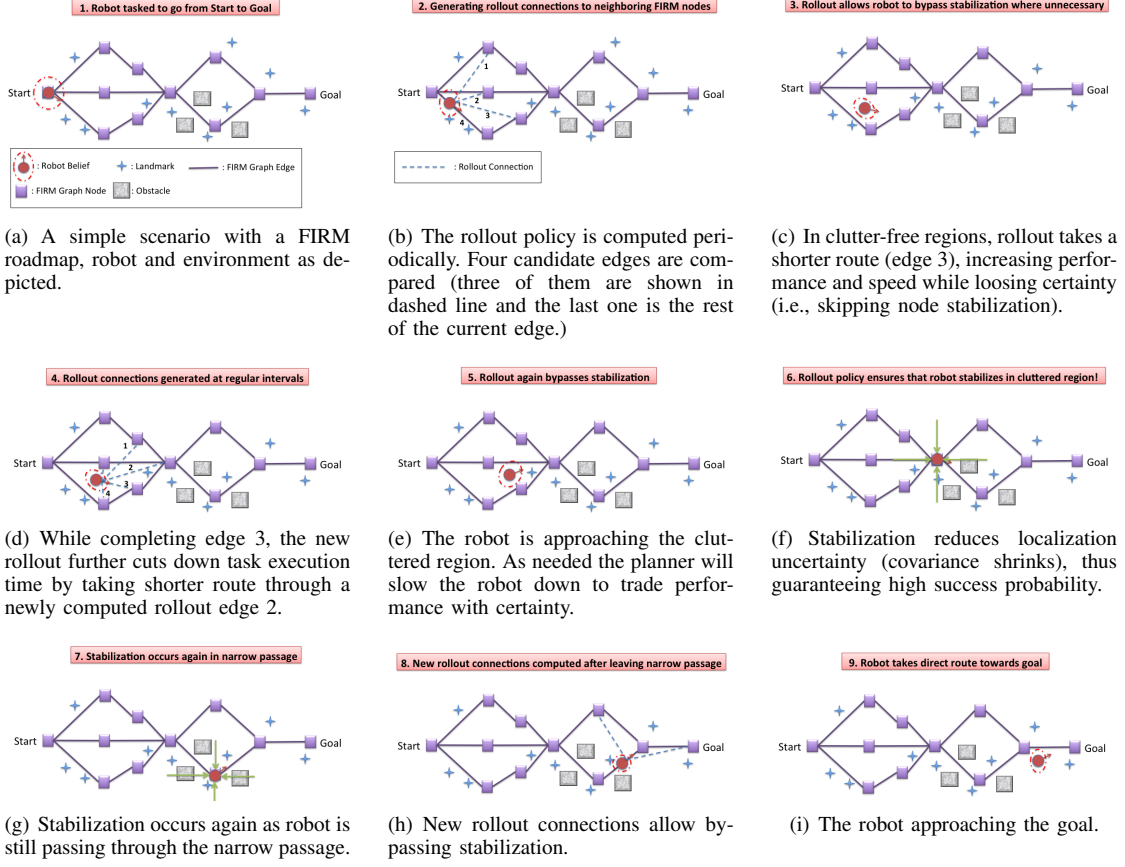


Fig. 2. A representational scenario depicting how rollout-based planner achieves higher performance compared to the standard FIRM algorithm while guaranteeing robustness. The 9 scenes depict different stages of task execution as the robot moves from the start to goal location.

Given a graph of these local controllers (Fig. 3(e)), we can define policy π^g on the graph as a mapping from graph nodes to edges; i.e., $\pi^g : \mathbb{V} \rightarrow \mathbb{M}$. Π^g denotes the set of all such graph policies. Having such a graph in belief space, we can form a tractable POMDP on the FIRM graph (so-called FIRM MDP):

$$\pi^{g*} = \arg \min_{\Pi^g} \mathbb{E} \sum_{n=0}^{\infty} C^g(B_n, \pi^g(B_n)) \quad (1)$$

where, B_n is the n -th visited node, and μ_n is the edge taken at B_n . $C^g(B, \mu) := \sum_{k=0}^{\infty} c(b_k, \mu(b_k))$ is the generalized cost of taking local controller μ at node B centered at b_0 . We incorporate the failure set in planning by adding a hypothetical FIRM node $B^0 = F$ to the list of FIRM nodes. Since the FIRM MDP in Eq.(1) is defined over a finite set of nodes, it can be solved by computing the graph node cost-to-go's through the following dynamic programming problem:

$$J^g(B^i) = \min_{\mu} \{C^g(B^i, \mu) + \sum_{\gamma=0}^N \mathbb{P}^g(B^{\gamma}|B^i, \mu) J^g(B^{\gamma})\} \quad (2)$$

and $\pi^g(B^i) = \mu^*$, where μ^* is the argument of above minimization. $\mathbb{P}^g(B^{\gamma}|B^i, \mu)$ is the probability of reaching B^{γ} from B^i under μ . The failure and goal cost-to-go's (i.e., $J^g(B^0)$ and $J^g(B^{goal})$) are set to a suitably high positive value and zero, respectively.

Collision (failure) probability of FIRM starting from a given

node B^i can be computed [5] as:

$$\mathbb{P}(Fail|B^i, \pi^g) = 1 - \Gamma_i^T (I - Q)^{-1} R_{goal}, \quad (3)$$

where, Γ_i is a column vector of zeros with only the i -th element set to one. Q is a matrix, whose (i, j) -th element is $Q[i, j] = \mathbb{P}(B^i|B^j, \pi^g(B^j))$ and R_{goal} is a column vector, whose j -th entry is $R_{goal}[j] = \mathbb{P}(B^{goal}|B^j, \pi^g(B^j))$. It can be shown that FIRM is an anytime algorithm [5], i.e., in a given static environment, increasing the number of nodes the cost (e.g., the failure probability) will go down. As will be discussed in the next section, this failure probability will be an upper bound for the failure probability of the FIRM-based rollout planner.

C. Concrete FIRM instance in our implementation

Here we discuss the concrete realization of the FIRM graph constructed for conducting the experiments.

One-step-cost: Although the objective function can be general, the cost function we use in our experiments includes the localization uncertainty, control effort, and elapsed time.

$$c(b_k, u_k) = \zeta_p \text{tr}(P_k) + \zeta_u \|u_k\| + \zeta_T. \quad (4)$$

where $\text{tr}(P_k)$ is the trace of estimation covariance as a measure of localization uncertainty. The norm of the control signal $\|u_k\|$ denotes the control effort, and ζ_T is present in the cost to penalize each time lapse. Coefficients ζ_p , ζ_u , and ζ_T are user-defined task-dependent scalars to combine these costs to achieve

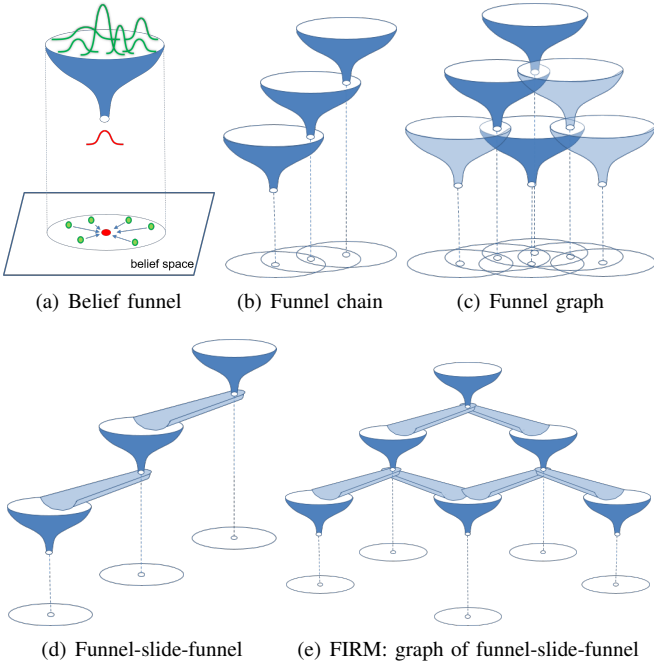


Fig. 3. An extension of sequential composition methods [17] to belief space. (a) A funnel in belief space that collapses a set of Gaussian distribution to a particular Gaussian distribution, referred to as the graph node or milestone. The 2D projection denotes the belief space, where each point represents a full Gaussian distribution. The projection of the mouth of funnel is a metaphor for its region of attraction in belief space. (b) A chain of funnels to guide the belief towards a goal. (c) A graph of funnels, where the tip of multiple funnels can fall into the region of attraction of a single funnel. (d) For a sparse set of funnels, one can use tracking controllers (slide) to create the funnel-slide-funnel structure. (e) Graph of funnel-slide-funnel. FIRM graph is of this type.

a desirable behavior. In the presence of constraints (such as obstacles in the environment), we assume the task fails if the robot violates these constraints (e.g., collides with obstacles). Therefore, collision and goal belief are terminal states such that $J^g(B^{goal}) = 0$ and $J^g(B^0) = J^g(F)$ is set to a suitably high cost-to-go. Note that typically the one-step-cost in Eq. 4 is defined in the state space (i.e., cost of taking action u at state s). While our cost can be written as a state space cost, writing it directly in belief space better demonstrates the active localization aspect of the work (in the sense of minimizing the uncertainty in the localization belief) along the plan.

Steering localization covariance: To construct a FIRM graph, we first need to sample a set of stabilizers (belief steering functions). Each stabilizer is a closed-loop controller, whose role is to drive the localization uncertainty (or belief) to a FIRM node. A stabilizer consists of a filter and a separated controller [37]. The filter governs the belief evolution and the separated-controller generates control signals based on the available belief at each time step [37]. To design these steering laws, we first sample a set of points $\mathcal{V} = \{\mathbf{v}^j\}$ in the robot's state space and then associated with each point we construct a stabilizer [6]. In the vicinity of each node \mathbf{v}^j , we rely on Stationary Kalman Filter (SKF) as the steering filter (which is constructed by linearizing the system about the target point \mathbf{v}^j) as the stabilizer's filter. It can be shown that for an observable system, the covariance under the j -th SKF approaches to covariance P_s^{+j} , which can be efficiently computed by solving a corresponding Discrete Algebraic Riccati Equation [8].

Steering localization mean: While steering belief toward

node B^j , separated-controller μ^{ij} is responsible for generating the control signals based on the available belief, i.e., $u_k = \mu^{ij}(b_k)$. The iRobot Create is a nonholonomic robot and is modeled as a unicycle (see Section V-A2), thus to steer the estimation mean toward the target node \mathbf{v}^j , one needs to use a controllers designed for stabilizing nonholonomic systems (e.g., [49], [47], [57]). However, the randomness of the estimation mean (resulted from randomness of observations) calls for a controller that can perform such stabilization under uncertainty. To this end, we implemented different controllers including polar coordinate-based controller [25] and Dynamic Feedback Linearization-based controller [4]. Observing the behavior of different controllers, we adopted a variant of the Open-Loop Feedback Control (OLFC) scheme [11] for stabilization purposes. In this variant of OLFC, for a given \mathbf{v}^j , we compute an open-loop control sequence starting from the current estimation mean and ending at \mathbf{v}^j . Then, we apply a truncated sequence of the first l controls ($l = 5$ in our experiments)¹. This process repeats every l steps until we reach the vicinity of \mathbf{v}^j .

FIRM graph: Associated with each sample \mathbf{v}^j , we can define the belief node $\hat{b}^j \equiv (\mathbf{v}^j, P_s^{+j})$. Defining FIRM node as a ball around this point $B^j = \{b : \|b - \hat{b}^j\| \leq \epsilon\}$, we can steer the Gaussian localization uncertainty to this ball with combination of OLFC and SKF. Accordingly, we sample N FIRM nodes $\{B^j\}_{j=1}^N$. The SKF/OLFC combination between nodes i and j forms the FIRM edge (local controller) and is denoted by μ^{ij} . We connect each node to k -nearest neighbors and the set of constructed edges is denoted by $\mathbb{M} = \{\mu^{ij}\}$. Then, we compute and store costs and transition probabilities associated with each edge by offline simulations. Finally, we solve the DP in Eq. (2) to get the optimal graph cost-to-go's $J^g(B^i)$ and policy $\pi^g(B^i)$ for all i .

FIRM requirements: It is worth noting that FIRM is not a general-purpose POMDP solver. It provides a solution for a class of POMDP problems (including SLAP) where one can design closed-loop controllers with a funneling behavior in belief space. In the current instantiation of the FIRM, designing funnels require knowledge about closed-form dynamics and sensor model. Also, the system needs to be locally linearizable at belief nodes, and the noise is assumed to be Gaussian.

III. SLAP VIA ROLLOUT-BASED DYNAMIC REPLANNING IN BELIEF SPACE

As discussed earlier, SLAP in this paper refers to the problem (re)planning dynamically every time the localization module updates the probability distribution on the robot's state. A principled solution to this problem, calls for resolving (or modifying) a POMDP in real-time. The SLAP problem in this paper corresponds to a restricted class of POMDPs, where the assumption follows the typical assumption in Kalman filter-based SLAM literature in robotics: discussions are limited to POMDPs where the state transition model and the observation model are given in the form of a locally linearizable (locally differentiable) explicit functions. The belief is Gaussian. The problem has a terminal belief and there is no discount factor.

To enable SLAP we resort to dynamic replanning in belief space which will handle changes in the environment and goal

¹Only one control (i.e., $l = 1$) is not enough due to the nonholonomicity of the system.

location, large deviations in the robot's location, and discrepancies between real and computational models. In this section, we discuss the extension of the RHC and Rollout policy [11] to the belief space to design a principled scheme for online replanning and increase the performance of FIRM by smart selective stabilization.

To make the connection with the rollout policy, we re-state the POMDP problem in a more general setting of the time-varying policy.

$$\begin{aligned} \pi_{0:\infty}(\cdot) &= \arg \min_{\Pi_{0:\infty}} \sum_{k=0}^{\infty} \mathbb{E}[c(b_k, \pi_k(b_k))] \\ \text{s.t. } b_{k+1} &= \tau(b_k, \pi_k(b_k), z_k), \quad z_k \sim p(z_k|x_k) \\ x_{k+1} &= f(x_k, \pi_k(b_k), w_k), \quad w_k \sim p(w_k|x_k, \pi_k(b_k)) \end{aligned} \quad (5)$$

In the above problem, we seek for a sequence of policies $\pi_{0:\infty} = \{\pi_0(\cdot), \pi_1(\cdot), \pi_2(\cdot), \dots\}$, where π_k maps any given b_k to the optimal action u_k , Π_k is the space of all possible policies at time step k , i.e., $\pi_k \in \Pi_k$. In the infinite horizon case, it can be shown that the solution is a stationary policy π_s , i.e., $\pi_1 = \pi_2 = \dots = \pi_s$ and the problem is reduced to the one introduced earlier in this paper. However, we keep the time-varying format for the reasons that will be clear further below.

As discussed earlier, solving the above POMDP problem is computationally intractable over continuous state, action, and observation spaces. However, the more difficult problem is to solve the SLAP problem which requires re-solving the above POMDP “online” every time the localization pdf is updated. We handle this problem by reusing computations in an efficient way as will be explained in the next subsection. However, we first start by RHC which is a natural way of thinking about such repeated online solutions.

RHC in belief space: Receding horizon control (often referred to as rolling horizon or model predictive control) was originally designed for deterministic systems [28] to cope with model discrepancies. For stochastic systems, where the closed-loop (feedback) control law is needed, formulation of the RHC scheme is up for debate [22, 43, 58, 70]. In the most common form of RHC [11] the stochastic system is approximated with a deterministic system by replacing the uncertain quantities with their typical values (e.g., maximum likelihood value.) In belief space planning the quantity that injects randomness in belief dynamics is the observation. Thus, one can replace the random observations z_k with their deterministic maximum likelihood value z_k^{ml} , where $z_k^{ml} := \arg \max_z p(z_k|x_k^d)$ in which x_k^d is the nominal deterministic value for the state that results from replacing the motion noise w by zero, i.e., $x_k^d = f(x_k^d, \pi_k(b_k^d), 0)$. The deterministic belief b^d is then used for planning in the receding horizon window. At every time step, the RHC scheme performs a two-stage computation. At the first stage, the RHC scheme for deterministic systems solves an open-loop control problem (i.e., returns a sequence of actions $u_{0:T}$) over a fixed finite horizon T as follows:

$$\begin{aligned} u_{0:T} &= \arg \min_{\mathbb{U}_{0:T}} \sum_{k=0}^T c(b_k^d, u_k) \\ \text{s.t. } b_{k+1}^d &= \tau(b_k^d, u_k, z_{k+1}^{ml}) \\ z_{k+1}^{ml} &= \arg \max_z p(z|x_{k+1}^d) \\ x_{k+1}^d &= f(x_k^d, u_k, 0) \end{aligned} \quad (6)$$

In the second stage, it executes only the first action u_0 and discards the remaining actions in the sequence $u_{0:T}$. However, since

the actual observation is noisy and is not equal to the z^{ml} , the belief b_{k+1} will be different that b_{k+1}^d . Subsequently, RHC performs these two stages from the new belief b_{k+1} . In other words, RHC computes an open loop sequence $u_{0:T}$ from this new belief, and this process continues until the belief reaches the desired belief location. Algorithm 1 recaps this procedure. State-of-the-art methods such as [52] and [67] utilize the RHC in belief space. [67] refers to the method as partially-closed loop RHC (PCLRHC) as it exploits partial information about future observations (i.e., z^{ml}) and does not ignore them.

Algorithm 1: RHC with most likely observations for partially-observable stochastic systems

```

1 input : Initial belief  $b_{current} \in \mathbb{X}$ ,  $B_{goal} \subset \mathbb{B}$ 
2 while  $b_{current} \notin B_{goal}$  do
3    $u_{0:T} =$  Solve the optimization in Eq.(6) starting from
      $b_0^d = b_{current}$ ;
4   Apply the action  $u_0$  to the system;
5   Observe the actual  $z$ ;
6   Compute the belief  $b_{current} \leftarrow \tau(b_{current}, u_0, z)$ ;

```

A known shortcoming of the stated RHC formulation is its limited horizon which might lead the system to local minima by choosing actions that guide the robot toward “favorable” states (with low cost) in the near future followed by a set of “unfavorable” states (with a high cost) in the long run. To improve the basic RHC, different variants have been proposed including the “rollout policy” [11]. Here, we discuss how they can be extended and realized in belief space.

Rollout policy in belief space: Another class of methods that aims to reduce the complexity of the stochastic planning problem in Eq.(5) is the class of rollout policies [11], which are more powerful than the described version of RHC in the following sense: First, they do not approximate the system with a deterministic one. Second, they avoid local minima using a suboptimal policy that approximates the true cost-to-go beyond the horizon. This function is referred to as the “base policy” and denoted by \tilde{J} . Formally, at each step of the rollout policy scheme, the following closed-loop optimization is solved:

$$\begin{aligned} \pi_{0:T}(\cdot) &= \arg \min_{\Pi_{0:T}} \mathbb{E} \left[\sum_{k=0}^T c(b_k, \pi_k(b_k)) + \tilde{J}(b_{T+1}) \right] \\ \text{s.t. } b_{k+1} &= \tau(b_k, \pi_k(b_k), z_k), \quad z_k \sim p(z_k|x_k) \\ x_{k+1} &= f(x_k, \pi_k(b_k), w_k), \quad w_k \sim p(w_k|x_k, \pi_k(b_k)) \end{aligned} \quad (7)$$

Then, only the first control law π_0 is used to generate the control signal u_0 and the remaining policies are discarded. Similar to the RHC, after applying the first control, a new sequence of policies is computed from the new point. The rollout algorithm is described in Algorithm 2.

Although the rollout policy in the belief space efficiently reduces the computational cost compared to the original POMDP problem, it is still formidable to solve since the optimization is carried out over the policy space. Moreover there should be a base policy that provides a reasonable cost-to-go \tilde{J} . In the following, we propose a rollout policy in the belief space based on the FIRM-based cost-to-go.

Algorithm 2: Rollout algorithm in belief space

```

1 input : Initial belief  $b_{current} \in \mathbb{B}$ ,  $B_{goal} \subset \mathbb{B}$ 
2 while  $b_{current} \notin B_{goal}$  do
3    $\pi_{0:T}$  = Solve optimization in Eq.(7) starting from
      $b_0 = b_{current}$ ;
4   Apply the action  $u_0 = \pi(b_0)$  to the system;
5   Observe the actual  $z$ ;
6   Compute the belief  $b_{current} \leftarrow \tau(b_{current}, u_0, z)$ ;

```

A. Enabling SLAP via FIRM-based rollout in belief space

In this section, we discuss how a rollout policy in belief space (and hence SLAP) can be realized using the FIRM framework. As explained briefly, in FIRM, the system transitions between two nodes² B^i and B^j at sampled beliefs b^i and b^j using a controller μ^{ij} . The global level decision-making is limited to when the system is in the region B^i and the rest of the time, the local controls are executed according to μ^{ij} . In FIRM-based rollout, we raise this limitation by forcing the system to globally replan at every step to enable SLAP. In particular, suppose that at time t , the belief state of the system is in b_t . Then we solve the following problem online for b_t :

1) We connect b_t to all it's FIRM neighbors in some radius R using suitable controllers μ^{tj} , designed in a similar way to the ones used as FIRM edges.

2) We evaluate the transition costs $C(b_t, \mu^{tj})$ and the probability of landing in nodes B^γ under the influence of the controller μ^{tj} at b_t , i.e., $\mathbb{P}(B^\gamma|b_t, \mu^{tj})$.

3) We evaluate the best edge outgoing from b_t by solving:

$$j^* = \arg \min_j \{C(b_t, \mu^{tj}) + \sum_{\gamma=0}^N \mathbb{P}(B^\gamma|b_t, \mu^{tj}) J^s(B^\gamma)\} \quad (8)$$

where $J^s(B^\gamma)$ is the nominal cost-to-go under the FIRM policy from node B^γ and $J^s(B^0)$ is the failure cost-to-go as discussed in Section II-B.

4) We choose μ^{tj^*} as the local controller at b_t if the expected success probability exceeds the current one. In other words, if μ^{ij} is the current local controller at time t , we only switch to μ^{tj^*} if below condition holds:

$$\mathbb{E}[\text{success}|b_t, \mu^{tj^*}] > \mathbb{E}[\text{success}|b_t, \mu^{tj}] \quad (9)$$

where expected success probability is

$$\mathbb{E}[\text{success}|b_t, \mu^{t\alpha}] = \sum_{\gamma=1}^N \mathbb{P}(B^\gamma|b_t, \mu^{t\alpha}) P^{\text{success}}(B^\gamma) \quad (10)$$

and $P^{\text{success}}(B^\gamma)$ is the probability of success for reaching the goal from FIRM node B^γ under the nominal FIRM policy.

Algorithm 3 describes planning with the proposed rollout process. We split the computation to offline and online phases. In the offline phase, we carry out the expensive computation of graph edge costs and transition probabilities. Then, we handle pose deviations and the changes in start/goal location by repeated online replanning, while reusing offline computations.

Below, we discuss the how Algorithm 3 realizes a variant of Eq. 8 and extends the rollout policy methods [11] to belief space.

² In the cartoon in Fig. 3, it looks like B^j is the sole destination for μ^{ij} . But, in dense graphs the belief under μ^{ij} might be absorbed by a different funnel before reaching B^j . The summation over γ in the following equations takes that into account.

Algorithm 3: Rollout algorithm with FIRM as base policy

```

1 input : Initial belief  $b_t$  and goal belief region  $B_{goal}$ 
2 Construct a FIRM graph and store nodes  $\mathbb{V} = \{B^i\}$ ,
   edges  $\mathbb{M} = \{\mu^{ij}\}$ , Cost-to-go  $J^s(\cdot)$ , and Success
   probabilities  $P^{\text{success}}(\cdot)$ ; // offline phase
3 while  $b_t \notin B_{goal}$  // online phase
4 do
5   Find  $r$  neighboring nodes  $\mathbb{V}_r = \{B^i\}_{i=1}^r$  to  $b_t$ ;
6   Set  $B_t = \{b_t\}$ ,  $J(B_t) = \infty$ , and  $S = 0$ ;
7   forall the  $B^j \in \mathbb{V}_r$  do
8      $\mu^{tj}$  = Generate controller from  $b_t$  to  $B^j$ ;
9      $C(b_t, \mu^{tj}), \mathbb{P}(B^\gamma|b_t, \mu^{tj})$  = Simulate  $\mu^{tj}$  to
       compute transition probability and expected cost;
10    Compute the expected success  $\mathbb{E}[\text{success}|b_t, \mu^{tj}]$ ;
11    if  $\mathbb{E}[\text{success}|b_t, \mu^{tj}] \geq S$  then
12      Compute the candidate cost-to-go as
        $J^{cand} = C(b_t, \mu^{tj}) + \sum_{\gamma=0}^N \mathbb{P}(B^\gamma|b_t, \mu^{tj}) J^s(B^\gamma)$ ;
13      if  $J^{cand} < J(B_t)$  then
14         $J(B_t) = J^{cand}$  and  $S = \mathbb{E}[\text{success}|b_t, \mu^{tj}]$ ;
15         $\mu^{tj^*} = \mu^{tj}$ ;
16  Apply the action  $u_t = \mu^{tj^*}(b_t)$  to the system;
17  Get the actual measurement  $z_{t+1}$ ;
18  Compute the next belief  $b_t \leftarrow \tau(b_t, u_t, z_{t+1})$ ;
19  if user submits a new goal state  $\mathbf{v}^{goal}$  then
20     $B^{goal} \leftarrow$  Sample the corresponding FIRM node;
21    Add  $B^{goal}$  to the FIRM graph;  $\mathbb{V} \leftarrow \mathbb{V} \cup \{B^{goal}\}$ ;
22    Connect  $B^{goal}$  to its  $r$  nearest neighbors using
       edges  $\{\mu^{(i,goal)}\}$ . Also,  $\mathbb{M} \leftarrow \mathbb{M} \cup \{\mu^{(i,goal)}\}$ ;
23     $[J^s(\cdot), P^{\text{success}}(\cdot)] = \text{DynamicProgramming}(\mathbb{V}, \mathbb{M})$ ;

```

Following, the concepts and terminology in [11], here, nominal FIRM policy plays the role of the base policy. Accordingly, the cost-to-go of the FIRM policy is used to approximate the cost-to-go beyond the horizon. Given a dense FIRM graph, where nodes partition the belief space, i.e., $\cup_i B^i = \mathbb{B}$, then at the end of time horizon T , the belief b_{T+1} belongs to a FIRM node B with known cost-to-go. With a sparse FIRM graph, where nodes do not cover the entire belief space, we design local policies that drive the belief into a FIRM node at the end of horizon. However, since the belief evolution is random, reaching a FIRM node at deterministic time horizon T may not be guaranteed. Therefore, we propose a new variant of rollout, defining the horizon based on belief (instead of time).

$$\pi_{0:\infty}(\cdot) = \arg \min_{\tilde{\Pi}} \mathbb{E} \left[\sum_{k=0}^{\mathcal{T}} c(b_k, \pi_k(b_k)) + \tilde{J}(b_{\mathcal{T}+1}) \right]$$

$$\text{s.t. } b_{k+1} = \tau(b_k, \pi_k(b_k), z_k), \quad z_k \sim p(z_k|x_k)$$

$$x_{k+1} = f(x_k, \pi_k(b_k), w_k), \quad w_k \sim p(w_k|x_k, \pi_k(b_k))$$

$$b_{\mathcal{T}+1} \in \cup_j B^j, \quad (11)$$

where for $b_{\mathcal{T}+1} \in B^j$ we have

$$\tilde{J}(b_{\mathcal{T}+1}) = J^s(B^j) \quad (12)$$

and $\tilde{\Pi}$ is a restricted set of policies under which the belief will reach a FIRM node B^j in finite time. In other words, if $\pi \in \tilde{\Pi}$ and $\pi = \{\pi_1, \pi_2, \dots\}$, then for finite \mathcal{T} , we have $\mathbb{P}(b_{\mathcal{T}+1} \in \cup_j B^j | \pi) = 1$. Thus, the last constraint in Eq. (11) is

redundant and automatically satisfied for suitably constructed $\tilde{\Pi}$. Also, the FIRM-based cost-to-go $J^s(\cdot)$ plays the role of the cost-to-go beyond the horizon $\tilde{J}(\cdot)$ (Eq. (12)).

Note that based on Algorithm 3, we can provide guarantees on the performance of the proposed method. Before formally stating the results, recall that at each instance of rollout computation, the current belief b_t is added as a virtual node $B_t^{virtual}$ to the FIRM graph to generate the augmented FIRM graph G_t^a . A virtual node being defined as a temporary node with no incoming edges, which is removed from the graph as soon as the system departs its vicinity.

Proposition 1: The performance and success probability of the FIRM-Rollout policy is lower bounded by the nominal FIRM policy at any belief state during execution of the planner.

Proof 1: As discussed, to compute the rollout at time t , belief b_t is added to the FIRM graph as a virtual node, with no incoming edges that drive the system into it. Therefore, the dynamic programming solution remains unchanged. Thus, the optimal cost-to-go from the virtual node $B_t^{virtual}$ is given by the minimum of the sum of the rollout edge cost and the cost-to-go from the target of rollout edge, i.e.,

$$J(B_t^{virtual}) = \min_j \{C(b_t, \mu^{tj}) + \sum_{\gamma=0}^N \mathbb{P}(B^\gamma | b_t, \mu^{tj}) J^s(B^\gamma)\}$$

Since the current FIRM edge is one of edges over which the above minimization is carried out, the cost-to-go (performance) with rollout is strictly upper (lower) bounded by the nominal FIRM policy cost (performance). Furthermore, due to the check in Eq. (9), it can be further assured that the probability of success of the rollout policy is strictly greater than that of the FIRM policy in static environments.

Once the rollout is computed and the target node is chosen, the robot starts executing the controller μ^{tj^*} and leaves the vicinity of node B^t . This node then gets removed from the graph. Thus, it is called a virtual node. Further, it should be noted that as the robot moves on the virtual edge (edge from node $B_t^{virtual}$ to B^{j^*}), the rollout process is repeated which leads the robot to skip the belief stabilization as needed. Consequently, as the robot moves, due to rollout, it chooses actions which are never worse-off than the nominal FIRM policy. We refer the reader to Fig.2 for a visual explanation of the process. ■

Remark: If the desired factor was merely the success probability, one can ignore the cost-to-go comparison condition Algorithm 3 and only maximize the success probability.

In addition to improving the performance while not compromising on the safety, the rollout procedure is particularly helpful in handling the changes in the environment map. We discuss this aspect in the following section.

B. Complexity Analysis

In this section, we analyze the computational complexity of the offline and online phase of the proposed algorithm.

Offline phase: We assume the environment is a hypercube $[0, w]^d$. For constructing the offline policy on a k -regular graph with N nodes, we have to simulate kN edges offline. Let us denote the number of particles describing belief by n_b^{off} . Assuming a fixed velocity $V = 1$ on edges, and assuming simulations steps occur at every Δt seconds, the number of simulation calls (including collision checks) is $n_{coll} = \sum_{s=1}^{kN} n_b^{off} \Delta t^{-1} l_s$, where l_s is the length of the s -th edge.

Assuming a uniform distribution of the sampled points (in the sense of infinity norm) in the configuration space, the density of points is $\rho = Nw^{-d}$. Accordingly, the dispersion [32, 42] of the sampled points is $\delta = wN^{-d-1}$. Assuming all edges have equal length (in the l^∞ -norm sense), the edge length of the underlying PRM (over which FIRM has been built) is $l_s = \delta = w\sqrt[d]{N}^{-1}$.

$$n_{coll} = (n_b^{off} \Delta t^{-1}) w k N^{1-d-1} \quad (13)$$

Online phase: In the online phase, we connect each node to all nodes in the neighborhood of radius R (in infinity norm). Thus, the size of neighboring area for connection is R^d , which encompasses $R^d * \rho$ neighboring points. For $R = r\delta$, it will encompass r^d points. Thus, we have r^d new edge in the online phase. It can be shown that the length of $(i+1)^d - i^d$ of these edges is in the range $i\delta < edgeLength < (i+1)\delta$.

For all edge lengths between $i\delta < l_s = edgeLength < (i+1)\delta$, let's approximate l_s by $i^+\delta$ where $i \leq i^+ \leq i+1$. Then, the sum of the length of all new edges is:

$$L_s = \sum_{s=1}^d l_s = \sum_{i=1}^r \sum_{s=(i-1)^d+1}^{i^d} l_s = \delta \sum_{i=1}^r ((i^d - (i-1)^d - 1)i^+)$$

Let us denote the number of particles describing belief by n_b . The number of simulation calls (including collision checks) is:

$$n_{coll} = n_b \Delta t^{-1} L_s = n_b \Delta t^{-1} \delta \sum_{i=1}^r ((i^d - (i-1)^d - 1)i^+)$$

Upper/lower bounds on the number of collision checks can be obtained by setting i^+ to its upper and lower bounds, i.e., $i+1$ and i . To gain further insight on the complexity let's assume i^+ is a constant (i.e., all edge lengths are the same) and set it to its maximum value $i^+ = R\sqrt[d]{N}w^{-1}$. Then, the upper bound on collision checks n_{coll}^+ is:

$$\begin{aligned} n_{coll}^+ &= (n_b \Delta t^{-1} w N^{-d-1}) (R\sqrt[d]{N}w^{-1}) [(R\sqrt[d]{N}w^{-1})^d - R\sqrt[d]{N}w^{-1}] \\ &= n_b \Delta t^{-1} w^{-d} R^{d+1} N - n_b \Delta t^{-1} w^{-1} R^2 \sqrt[d]{N} \end{aligned} \quad (14)$$

Given this upper-bound on the computation time and given uniform grid sampling strategy, the online computation time grows sub-linearly with the number of underlying FIRM nodes N in the worst case. Also, for a given dimension the online computation time is polynomial in the connection radius R . To remove the dimension from equation and extend the results to random sampling, we can write the first term of the above equation as:

$$n_{coll}^+ = (n_b \Delta t^{-1}) R V \rho$$

where ρ is the density of samples in the environment and V is the volume of the connection neighborhood and R is the radius of the connection neighborhood.

C. Enabling SLAP in changing environments

In this section, we discuss the ability of the proposed planner to handle changes in the obstacle map. We focus on a challenging case, where changes in the obstacle map are persistent and can possibly eliminate a homotopy class of solutions. Doors are an important example of this class. If the robot observes a door is closed (which was expected to be open), it might have

to globally change the plan to get to the goal from a different passage. This poses a challenge to the state-of-the-art methods in the belief space planning literature.

To handle such changes in the obstacle map and replan accordingly, inspired by the lazy evaluation methods for PRM frameworks [13], we propose a method for lazy evaluation of the generated feedback tree, referred to as “lazy feedback evaluation” algorithm. The basic idea is that at every node the robot re-evaluates *only* the next edge (or the next few edges up to a fixed horizon) that the robot will most likely take. By re-evaluation, we mean it needs to re-compute the collision probabilities along these edges. If there is a significant change (measured by α in Alg. 4) in the collision probabilities, the dynamic programming problem is re-solved and new cost-to-go’s are computed. Otherwise, the cost-to-go’s remain unchanged and the robot keeps following its rollout-policy. Such lazy evaluation (computing the collision probabilities for a single edge or a small number of edges) can be performed online. The method is detailed in Algorithm 4.

Algorithm 4: Lazy Feedback Evaluation (Lazy Replanning)

```

1 input : FIRM graph
2 output : Updated cost-to-go,  $J^g(\cdot)$  and success
   probabilities  $P^{success}(\cdot)$ 
3 Perceive the obstacles map;
4 if there is a change in map then
5    $\mathcal{F} \leftarrow$  Retrieve the sequence of nominal edges
   returned by feedback up to horizon  $l$ ; Set  $ctr = 1$ ;
6   forall the edges  $\mu \in \mathcal{F}$  do
7     Re-compute collision probabilities  $\mathbb{P}_{new}(B, \mu)$ 
     from start node  $B$  of edge  $\mu$ ;
8     if  $|\mathbb{P}_{new}(B, \mu) - \mathbb{P}(B, \mu)| > \alpha$  then
9        $\mathbb{P}(B, \mu) \leftarrow \mathbb{P}_{new}(B, \mu)$ ;
10       $ctr++$ 
11   Update edge set  $\mathbb{M}$  based on new transition
   probabilities;
12   if  $ctr > 0$  then
13      $[J^g(\cdot), P^{success}(\cdot)] = \text{DynamicProgramming}(\mathbb{V}, \mathbb{M})$ ;
14 return  $J^g(\cdot)$  and  $P^{success}(\cdot)$ ;
```

Remark: Another challenge with these persistent changes is that they stay in the memory. Imagine a case where the robot is in a room with two doors. Suppose after checking both doors, the robot realizes they are closed. In those cases where there is no homotopy class of solutions to the goal, the door state is reset to “open” after a specific amount of time to persuade the robot to recheck the state of doors.

It is important to note that it is the particular structure of the proposed planner that makes such replanning feasible online. The graph structure of the underlying FIRM allows us to *locally* change the collision probabilities in the environment without affecting the collision probability of the rest of the graph (i.e., properties of different edges on the graph are independent of each other; see Fig. 3 and 1). Such a property is not present in the state-of-the-art sampling-based belief space planners (e.g., [54],[68]), where the collision probabilities and costs on *all* edges are dependent on each other and hence need to be re-

computed.

IV. SIMULATION RESULTS

In this section, we present simulation results for a comparison of the standard FIRM algorithm versus FIRM with Rollout in a 2D navigation problem. The simulation represents a motion planning scenario wherein the robot is tasked to go from a start location to multiple goal locations sequentially in an environment with clutter, narrow passages and assymmetrically placed landmarks/beacons. We conduct two simulations, the first with the standard FIRM algorithm and the second with the Rollout based policy framework presented in this work. All simulations were carried out on a Dell Precision Desktop with a Quad-Core Intel(R) Xeon(R) E5-2609 CPU running at 2.40GHz and 16GB of RAM running Ubuntu 14.04.

1) *Motion Model:* We represent the motion of the robot by a simple planar kinematic model in which each component of the state can be independently controlled. The state of the robot at time k is denoted by $x_k = (x_k, y_k, \theta_k)^T$ (2D position and the heading angle). We denote the control as $u_k = (v_{x,k}, v_{y,k}, \omega_k)^T$ and the process noise by $w_k = (n_{v_x}, n_{v_y}, n_{\omega})^T \sim \mathcal{N}(0, \mathbf{Q}_k)$. Let f denote the kinematics of the robot such that $x_{k+1} = f(x_k, u_k, w_k) = x_k + u_k \delta t + w_k \sqrt{\delta t}$.

2) *Observation Model:* We use a range-bearing landmark based observation model in which the landmarks are assumed to be passive beacons placed at various locations in the environment. Each landmark has a unique ID associated to it and this ID can be read by the sensor along with the range and bearing relative to the robot. Let ${}^i\mathbf{L}$ be the location of the i -th landmark, then the displacement vector ${}^i\mathbf{d}$ from the robot to ${}^i\mathbf{L}$ is given by ${}^i\mathbf{d} = [{}^i d_x, {}^i d_y]^T := {}^i\mathbf{L} - \mathbf{p}$, where $\mathbf{p} = [x, y]^T$ is the position of the robot. Therefore, the observation ${}^i z$ of the i -th landmark can be described as ${}^i z = {}^i h(x, {}^i v) = [\|{}^i\mathbf{d}\|, \text{atan2}({}^i d_y, {}^i d_x) - \theta]^T + {}^i v$.

The observation noise is assumed to be zero-mean Gaussian such that ${}^i v \sim \mathcal{N}(\mathbf{0}, {}^i\mathbf{R})$ where ${}^i\mathbf{R} = \text{diag}((\eta_r \|{}^i\mathbf{d}\| + \sigma_b^r)^2, (\eta_\theta \|{}^i\mathbf{d}\| + \sigma_b^\theta)^2)$. The measurement quality decreases as the robot gets farther from the landmarks and the parameters η_r and η_θ determine this dependency. σ_b^r and σ_b^θ are the bias standard deviations.

3) *Environment and Scenario:* The environment in Fig. 4(a) represents a 2D office space measuring $21\text{m} \times 21\text{m}$ with obstacles and beacons placed in it. The robot is represented by a circular disk of radius 1m. There are two narrow passages $P1$ and $P2$ which represent paths or edges of low transition probability/high collision probability. The narrow passages are 1.25m wide thus offering a very small clearance for the robot to pass through. The robot is placed at starting location ‘A’ and tasked to visit 4 different locations in a sequential manner, these are marked as B, C, D and E in the environment. Thus, we split the task into 4 sequential segments: 1) $A \rightarrow B$, 2) $B \rightarrow C$, 3) $C \rightarrow D$, and 4) $D \rightarrow E$. The simulation is carried out twice, once with the standard FIRM algorithm and once with the proposed rollout-based method in belief space. In the following sections, we explain each simulation and then present a comparative analysis.

4) *Planning with Standard FIRM Algorithm:* Here we give a brief explanation of how the FIRM algorithm works and introduce some key terminology and then explain the individual steps of the simulation itself.

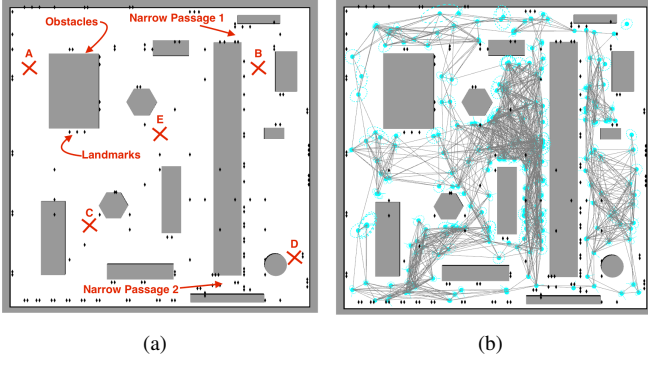


Fig. 4. (a) The simulation environment. The black diamonds depict the landmarks, the grey polygons are the obstacles and the white space represents the free space. The locations of interest that the robot is tasked to visit are marked by red crosses. The two narrow passages P1 and P2 are marked, these represent regions of high collision probability (risky) due to the small clearance. (b) The underlying FIRM roadmap, the grey lines depict edges and the cyan disks are the nodes, the dashed ellipses represent the stationary covariance of the FIRM nodes.

Offline-phase: First, we construct the underlying FIRM roadmap as depicted in Fig. 4(b). This roadmap is constructed by uniformly sampling configurations in the free space and then building the node beliefs on these configurations which are the nodes of our FIRM roadmap. To create each belief node, we follow the procedure in Section II-C. In short, we linearize the system dynamics and sensor model around the sampled configuration point. We create the stationary Kalman Filter corresponding to this local linear system and find its reachable belief by solving the corresponding Riccati equation. At each node, there exists a stabilization controller (stabilizer) which drives beliefs from a region around the node to the stationary belief. The edges of the FIRM roadmap are generated by first finding valid (collision free) straight line connections between neighboring nodes (nodes within a neighborhood of fixed radius R) and then generating edge controllers which drive the belief from the start belief node to the vicinity of the goal belief node. For each edge in the graph, we run Monte Carlo simulations to compute the expected execution cost and transition probability. Once we have constructed the underlying FIRM roadmap, we store it for use in the online phase.

Online-phase: In the online phase, the planner has access to the stored roadmap that is computed offline and receives a start and a goal configuration. These configurations are added to the existing roadmap by computing the appropriate stationary belief, stabilizer and edge controllers. Now, using the fact that FIRM preserves the optimal sub-structure property (edges are independent of each other; see Fig. 3 and 1), we solve the Dynamic Programming on the graph for the given goal location. Before we proceed, we define a few terms that will be used frequently in the subsequent text:

- **FIRM Feedback Tree:** The solution of the dynamic programming problem, i.e., π^g , is visualized with a *feedback tree*. Recall that π^g is a mapping (look-up table) that returns the next best edge for any given graph node. Therefore, for each node, the feedback tree contains only one outgoing edge ($\mu = \pi^g(B^i)$) that pulls the robot towards the goal. The feedback tree is rooted at the goal node.
- **Most-Likely Path (MLP):** The most likely path is generated by concatenating the sequence of edges to take for the motion plan concerning the given start/goal combination i.e. beginning

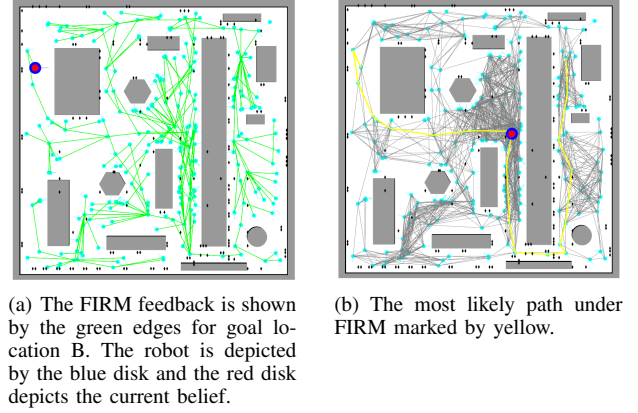


Fig. 5. Phase 1 of policy execution with FIRM, starting at A and going to B. The locations A and B are marked in Fig. 4(a)

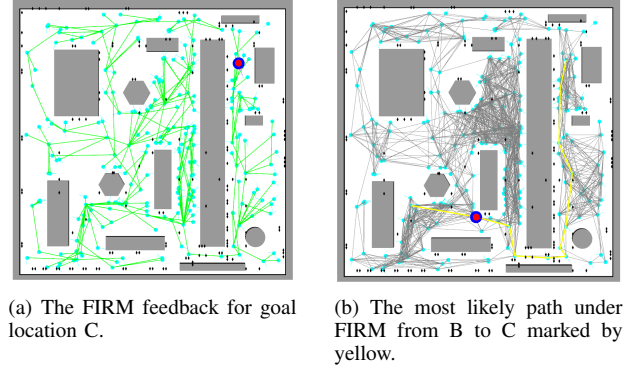
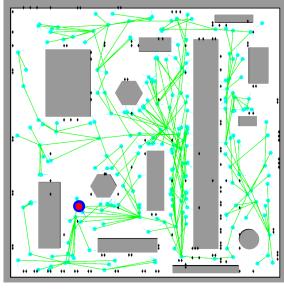


Fig. 6. Phase 2 of policy execution with FIRM, starting at B and going to C. The locations B and C are marked in Fig. 4(a)

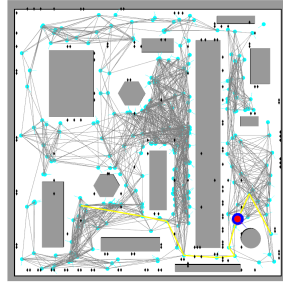
at the start node and then adding the subsequent edges as given by the FIRM feedback. Thus, it depicts the solution as a path which is instructive to us. Note that the robot does not follow the most-likely path exactly due to the noises.

In segment 1, the planner is given A and B as the start and goal locations respectively and computes the FIRM Feedback, Fig. 5(a) shows the feedback on the graph. Once the feedback is computed, the robot starts executing the feedback policy. We see in Fig. 5(b) that the robot is close to the most-likely path. Note that the robot does not exactly follow the most-likely path due to noise, however in this case, the noise is not high enough to change the homotopy class. Figure 5(b) shows this path passing through the narrow passage P2. On its way to the goal, the robot follows the edge controllers returned by the feedback policy and stabilizes to the FIRM nodes at the end of the edges. Once the robot reaches the first goal location B, we begin segment 2 and the planner is given the new goal location C. A new feedback policy is computed online (Fig. 6(a)) and the robot moves to C along the MLP (Fig. 6(b)), again passing through the narrow passage P2. Similarly, a new feedback policy is computed for each subsequent goal (Fig. 7(a) and Fig. 8(a)) and the robot eventually reaches its final destination E successfully.

5) Planning with rollout-based planner: Here again, we first begin with the underlying FIRM roadmap that is constructed offline as explained in the previous section and then find the FIRM feedback in segment 1 for start and goal locations A and B respectively. Once the feedback policy is computed (same as

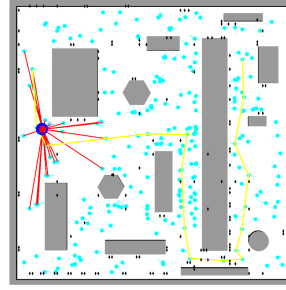


(a) The FIRM feedback for goal location D.

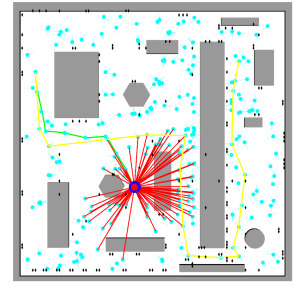


(b) The most likely path under FIRM from C to D (yellow).

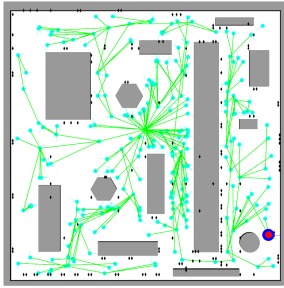
Fig. 7. Phase 3 of policy execution with FIRM, starting at C and going to D.



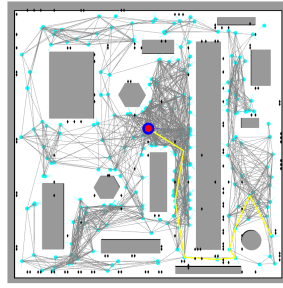
(a) The robot checks for a new rollout policy by locally checking connections with neighbors. The rollout connections are shown in red. The yellow path depicts the most-likely path under the nominal FIRM feedback.



(b) Rollout guides the robot away from the MLP through a shorter path through the relatively open area.



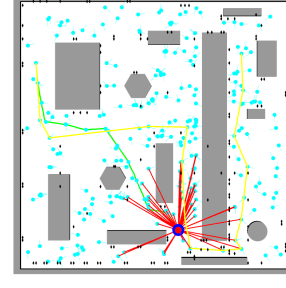
(a) The FIRM feedback for goal location E.



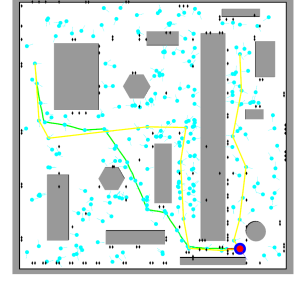
(b) The most likely path under FIRM from D to E (yellow).

Fig. 8. Phase 4 of FIRM policy execution, starting at D and going to E.

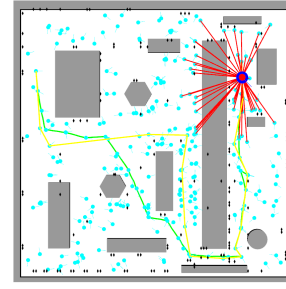
that in Fig. 5(a)), the rollout-based planner starts by following the feedback tree. However, from here on the rollout behavior emerges. Let the rollout update time interval be defined as $T_{rollout}$. Thus, at every $T_{rollout}$ seconds the planner locally computes connections to existing FIRM nodes in a neighborhood of radius R centered at the robot's belief, i.e., the planner locally generates edge controllers with their associated cost-to-connect and the transition probability. Since FIRM gives us the cost-to-go to the goal from each and every FIRM node, by finding the local connection costs, the planner checks which connection provides the lowest sum of the cost to connect to a neighborhood node and the cost-to-go from that node (Eq. 8). The connection with the lowest sum is chosen as the next edge to follow. Fig. 9(a) shows the planner checking connections (red-edges) locally to neighboring FIRM nodes. An important behavior emerges in segment 1, as the robot proceeds, the rollout is able to find a shorter path through the relatively open area by skipping unnecessary stabilizations (as shown in Fig. 9(b) and 9(c)). As the robot traverses the narrow passage, the rollout changes the behavior by forcing the robot to “stabilize” to the node as it concludes it is an optimal decision to further reduce uncertainty while proceeding through the narrow passage (shown in Fig. 9(d)). Eventually the robot reaches location B through the path as marked in green in Fig. 9(f). It is clear that the rollout gives the robot a distinct advantage over the nominal FIRM plan as it guides the robot through a direct, much shorter route. Further, it should be noted that although the end segments of the two paths (i.e. after exiting the narrow passage) look similar, they differ significantly in the velocity profiles. Along the yellow path, the robot stabilizes to each and every FIRM node along the way while along the green path (rollout), it skips stabilizations and



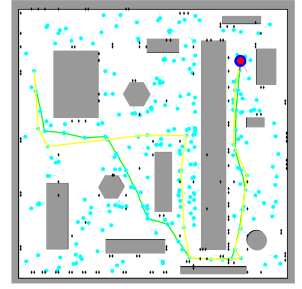
(c) Robot approaches narrow passage 2 through a more direct path as compared to the MLP.



(d) The robot stabilizes at a FIRM node while passing through the narrow passage.



(e) The robot approaches goal location B.



(f) The path taken under the rollout (green) is geometrically shorter than the path under FIRM. Further, by skipping unnecessary stabilizations, the robot moves faster.

Fig. 9. Phase 1 with rollout: Starting at A and going to B.

only stabilizes when necessary. This helps the robot maintain a higher average velocity while executing the plan.

At point B, a new FIRM feedback is computed online for the next goal location C (segment 2). Prior to approaching the narrow passage, rollout drives the robot to stabilize to a FIRM node as seen in Fig. 10(a). This stabilization helps the robot become more certain of its state before passing through a region with low transition probability. The robot then passes through the narrow passage P2 and reaches C through the path (green) shown in Fig. 10(b). In segment 3, (goal location D), the plan again takes the robot through the narrow passage P2, and during this segment, the planner drives the robot to stabilize to three FIRM nodes on the way to D. (Fig. 11 shows segment 3).

Finally, in segment 4, the robot is given goal location E. Here, we notice that while returning to the narrow passage from D,

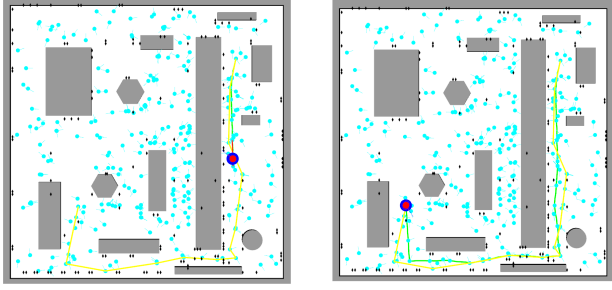


Fig. 10. Phase 2: Starting at B and going to C.

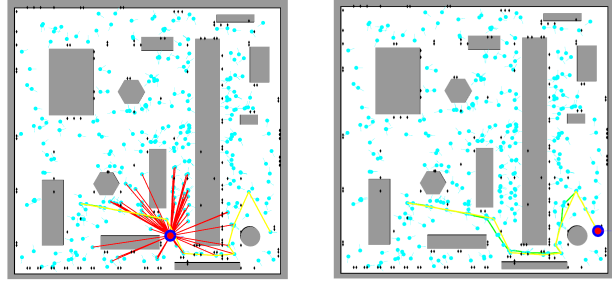


Fig. 11. Phase 3: Starting at C and going to D.

the rollout causes the robot to divert slightly from the MLP as marked in Fig. 12(a). The same was not observed when the robot was moving towards D. The reason that this happens is that the cost to reach D through the MLP was lower than the computed cost through a shorter but riskier path. However, on the way to E, while moving away from D, the MLP has a higher cost than the one given by a more direct path as given by rollout. Finally, the mission is completed and the robot arrives at E as shown in Fig. 12(b).

6) *Analysis of Simulation Results:* In this section, we present the analysis for the simulation results from the previous section by running the planner many times. The key finding is that using rollout based FIRM policy execution significantly increases the performance of the standard FIRM implementation while preserving its robustness and scalability.

Cost of Execution: We have recorded the amount of uncer-

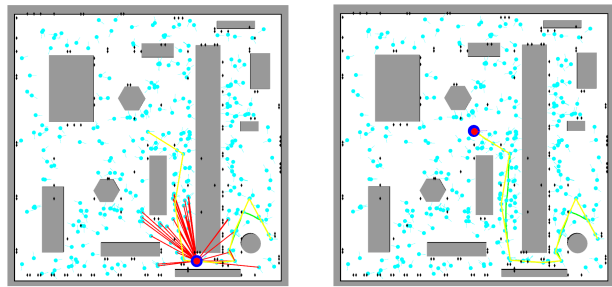


Fig. 12. Phase 4: Starting at D and going to E.

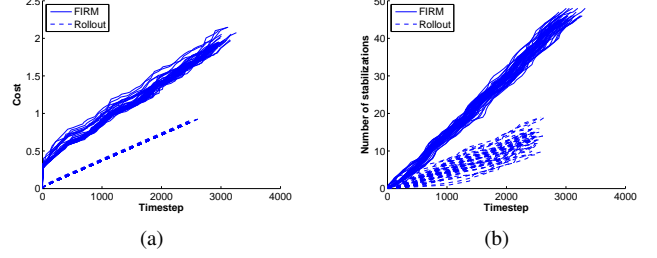


Fig. 13. Performance comparison of the original FIRM algorithm and rollout-based planner on 50 runs. (a) Cost of execution: The execution cost for FIRM rises faster than the cost of rollout based policy. (b) The number of belief nodes that the robot stabilizes to, during plan execution, which is consistently lower for the rollout-based planner.

tainty (trace of covariance) along the robot's path. Figure 13(a) shows the accumulative version of this cost on 50 runs for the same task using rollout-based planner and with standard FIRM. We can see that the cost for the rollout based policy rises slower than the cost for FIRM, and as the planning horizon increases, rollout offers increasing returns in performance.

Stabilization to FIRM Nodes: One of the key performance issues for the standard FIRM algorithm is also one of its major strengths, i.e., the node stabilization process of reaching nodes in belief space. Node stabilization makes FIRM robust and scalable while maintaining the optimal sub-structure of the graph (all the edges are independent of each other; see Fig. 3). Thus, though stabilization allows FIRM to provide certain guarantees, it might lead to slower robot motion in general as it needs to reach each belief node along the way, thus increasing the time to complete the task and also adding cost during the stabilization process at each node. Rollout-based planner brings a higher level of intelligence to this process of node stabilization. Rollout performs stabilization as and when required and bypasses it when possible. Thus, by bypassing the stabilization when not required, rollout allows the robot complete the task faster as well as with less execution cost. Fig.13(b) shows the number of nodes the robot has stabilized to with the passage of time in 50 runs. In this example, the robot stabilizes to ~ 45 nodes under FIRM compared to ~ 10 nodes under rollout-based planner ($\sim 75\%$ reduction), while the difference is growing as the task becomes longer.

Time of Execution: Task completion Another indicators of performance is how much time a planner takes to complete the task while guaranteeing a high likelihood of success. From Fig. 13(a) and 13(b), the time taken to complete the task with rollout is around 2500 timesteps (250 seconds) compared to 3000 timesteps (300 seconds) for FIRM. There is around a 15% reduction in the time to complete the task compared to the standard FIRM algorithm. The improvement in execution time makes the rollout-based planner a better candidate than FIRM for time-sensitive applications.

V. EXPERIMENTAL RESULTS FOR A PHYSICAL SYSTEM

This section includes the main contribution of this paper: demonstrating an online POMDP-based solution for the simultaneous localization and planning problem on a physical robot in a real-world setting. We discuss the details of implementation and report the important lessons learned while investigating the application of FIRM-based SLAP on physical systems. A video demonstrating the system is available in [1].

A. Target Application and System Set Up

Consider a scenario, where the robot needs to operate and reach a goal in an office environment. Each time the robot reaches a goal, a new goal is submitted by a higher-level application (e.g., manually by a user or different users). During a set of long runs, we investigate the performance of online-replanning in SLAP and the robustness of the method to (i) changing obstacles, such as doors, and moving people, (ii) changes in the goal location, (iii) deviations due to intermittent sensory failures, and (iv) kidnap situations (significant sudden deviation in the robot's location). During the run, there are many situations where the robot needs to replan: It needs to replan each time a new goal is submitted and move toward the new goal. Also, the robot encounters changes in the obstacle map. For example the robot may expect a door to a room be open while planning a path, but senses that it is closed on reaching there. Similarly, it encounters moving people and must avoid bumping into them. Observing these changes, the robot updates its map and replans online to update its policy. Moreover, the robot might be deviated from its nominal path. As an extreme case of this situation, we look into the "kidnapped" situation, where a person might move the robot to an unknown location during the run. Thus, the robot needs to recover from this catastrophic situation. Finally, the robot may deviate from its nominal location due to temporary failures in the sensing system. In all these cases an online replanning scheme can help the robot to recover from the aforementioned situations and accomplish its goal. It is worth noting that the main focus of all the different experiments in this section is to demonstrate the performance and feasibility of SLAP by enabling online belief space planning on physical robots.

1) *Environment*: The specific environment for conducting our experiments is the fourth floor of the Harvey Bum Bright (HRBB) building at the Texas A&M University campus in College Station, TX. The floor-plan is shown in Fig. 14. The floor spans almost 40 meters of hallways whose average width is approximately 2 meters, which is distinguished in yellow and blue in Fig. 14. The particular set of experiments reported in this paper is conducted in the region which is highlighted in blue in Fig. 14, part of which contains a large cluttered office area (407 area). This area has interesting properties that makes the planning more challenging: (i) 407 area is highly cluttered inside and includes lots of chairs and more than 15 workstations. (ii) As is seen in Fig. 14, there are several doors in this area which may be opened or closed. Two of these doors (front-door and back-door) are labeled in Fig. 14. (iii) There are objects such as chairs and trash-cans in this environment which usually get displaced. (iv) There are moving people, who are avoided using a reactive behavior, which may displace the robot from its planned path. This further introduces challenges for the high level planner.

2) *Robot Model*: The physical platform utilized in our experiments is an iRobot Create mobile robot (See Fig. 15). The robot can be modeled as a unicycle whose kinematics are as follows:

$$x_{k+1} = f(x_k, u_k, w_k) = \begin{pmatrix} x_k + (V_k \delta t + n_v \sqrt{\delta t}) \cos \theta_k \\ y_k + (V_k \delta t + n_v \sqrt{\delta t}) \sin \theta_k \\ \theta_k + \omega_k \delta t + n_\omega \sqrt{\delta t} \end{pmatrix}, \quad (15)$$

where $x_k = (x_k, y_k, \theta_k)^T$ describes the robot state, in which $(x_k, y_k)^T$ is the 2D position of the robot and θ_k is the heading angle of the robot, at time step k . Control commands are the



Fig. 14. Floor-plan of the environment, in which experiments are conducted. linear and angular velocities $u_k = (V_k, \omega_k)^T$. We use the Player robot interface [29] to send these control commands to the robot.

Motion noise: The motion noise vector is denoted by $w_k = (n_v, n_\omega)^T \sim \mathcal{N}(0, \mathbf{Q}_k)$, which mostly arose from uneven tiles on the floor, wheel slippage, and inaccuracy in the length of time control signals need to be applied. Experimentally, we found that in addition to the fixed uncertainty associated with the control commands there exists a portion of the noise that is proportional to the signal strength. Thus, we model the variance of the process noise at the k -th time step as $\mathbf{Q}_k = \text{diag}((\eta V_k + \sigma_b^V)^2, (\eta \omega_k + \sigma_b^\omega)^2)$, where in our implementations we have $\eta = 0.03$, $\sigma_b^V = 0.01\text{m/s}$, $\sigma_b^\omega = 0.001\text{rad}$.



Fig. 15. A picture of the robot (an iRobot Create) in the operating environment. Landmarks can be seen on the walls.

3) *Sensing Model*: For sensing purposes, we use the on-board camera existing on the laptop. We perform a vision-based landmark detection based on ArUco (a minimal library for Augmented Reality applications) [46]. Each landmark is a black and white pattern printed on a letter-size paper. The pattern on each landmark follows a slight modification of the Hamming code, and has a unique id, so that it can be detected robustly and uniquely. Landmarks are placed on the walls in the environment (see Fig. 15). The absolute position and orientation of each landmark in the environment is known. The ArUco library performs the detection process and presents the relative range and bearing to each visible landmark along with its id. Therefore, if we denote the j -th landmark position in the 2D global coordinates as jL , we can model the observation as a range-bearing sensing system:

$${}^jz_k = [||{}^jd_k||, \text{atan2}({}^jd_{2k}, {}^jd_{1k}) - \theta]^T + {}^jv, \quad {}^jv \sim \mathcal{N}(\mathbf{0}, {}^j\mathbf{R}),$$

where ${}^jd_k = [{}^jd_{1k}, {}^jd_{2k}]^T := [\mathbf{x}_k, \mathbf{y}_k]^T - {}^jL_j$.

Measurement noise: A random vector j_v models the measurement noise associated with the measurement of the j -th landmark. Experimentally, we found that the intensity of the measurement noise increases by the distance from the landmark and by the incident angle. The incident angle refers to the angle between the line connecting the camera to landmark and the wall, on which landmark is mounted. Denoting the incident angle by $\phi \in [-\pi/2, \pi/2]$, we model the sensing noise associated with the j -th landmark as a zero mean Gaussian, whose covariance is

$${}^j\mathbf{R}_k = \text{diag}((\eta_{r_d} \| {}^j\mathbf{d}_k \| + \eta_{r_\phi} |\phi_k| + \sigma_b^r)^2, (\eta_{\theta_d} \| {}^j\mathbf{d}_k \| + \eta_{\theta_\phi} |\phi_k| + \sigma_b^\theta)^2), \quad (16)$$

where, in our implementations we have $\eta_{r_d} = 0.1$, $\eta_{r_\phi} = 0.01$, $\sigma_b^r = 0.05\text{m}$, $\eta_{\theta_d} = 0.001$, $\eta_{\theta_\phi} = 0.01$, and $\sigma_b^\theta = 2.0\text{deg}$.

Full vector of measurements: At every step the robot observes a subset of the landmarks, which fall into its field of view. Suppose at a particular step the robot can see r landmarks $\{L_{i_1}, \dots, L_{i_r}\}$. The concatenation of visible landmarks is the total measurement vector that is denoted by $z = [{}^{i_1}z^T, \dots, {}^{i_r}z^T]^T$ and due to the independence of measurements of different landmarks, the observation model for all landmarks can be written as $z = h(x) + v$, where $v = [{}^{i_1}v^T, \dots, {}^{i_r}v^T]^T \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, where $\mathbf{R} = \text{diag}({}^{i_1}\mathbf{R}, \dots, {}^{i_r}\mathbf{R})$.

B. SLAP versus Regular Localization and Planning

In this section, we contrast the results of a regular localization and planning with the proposed SLAP solution. Regular localization and planning here refers to a method, where first the planner (ignoring the localizer) generates a path for the robot to follow. Then, in the execution phase, we run the localization algorithm to compute the mean and follow the pre-computed trajectory using a closed-loop controller. However, in the proposed SLAP solution, the planner takes the localizer into account in the planning phase and replans simultaneously as the localizer updates its estimation.

The environment is shown in Fig. 16. Blue regions are obstacles and black regions are free space. Landmarks are shown by small white diamonds. The start and goal locations for the motion planning problem are marked in Fig. 16. The goal location is inside room 407 (see Fig. 14) and the start is close to the front door.

Regular planning and localization: To select a suitable planner, we tried a variety of traditional planners such as PRM, RRT, and their variants. We observed that due to the motion noise of this low-cost robot and sparsity of the information in the certain parts of this environment, many of these variants lead to collisions with obstacles and cannot reach the goal point. The best results, we got was from the MAPRM (Medial-Axis PRM) method [71]. This planner is computationally more expensive than the other variants but is very powerful in dealing with collisions. since it samples points and construct a PRM that has the most clearance from obstacles, so it leads to plans with minimum collisions in our experiments. An MAPRM graph (in white) for this environment is shown in Fig. 16.

As can be seen, there exist two homotopy classes of paths between the start and goal nodes: Through the front door of room 407 and through the back door of the room. From Fig. 16, it is obvious that the path through the front door is shorter. Moreover, the path through the front door has a larger obstacle

clearance (larger minimum distance from obstacles along the path) compared to the path through the back door (since the back door is half open). Therefore, based on conventional metrics in deterministic settings, such as shortest path or maximum clearance, MAPRM chooses the path through the front door over the path through the back door. The feedback tree that results from solving the DP in this case is shown in Fig. 17. As expected, the DP guides the robot to go through the front door.

To execute the plan generated from PRM, we use time-varying LQG controllers to keep the robot close to the nominal path returned as the solution of the PRM-based planning. However, due to the lack of enough information along the nominal path, the success rate of this plan is low, and the robot frequently collides with obstacles along the path as the robot is prone to drift. The success probability along the nominal path is computed by a multiple (100) runs and is equal to 27% (27 runs out of 100 runs were collision-free).

FIRM-based SLAP: As can be seen in Fig. 16, the distribution of information is not uniform in the environment. The density of landmarks (information sources) along the path through the back door is more than the landmarks along the path through the front door. In the FIRM-based SLAP, though, the landmark map is automatically incorporated in the planning phase in a principled way. As a result, it leads to a better judgment of how narrow the passages are. For example, in this experiment, although the path through the front door is shorter than the path through the back door, considering the information sources, the success probability of going through the back door is much more than going through the front door. Such knowledge about the environment is reflected in the FIRM cost-to-go and success probability in a principled framework. As a result, it generates a policy that suits the application, taking into account the uncertainty, and available information in the environment. Solving DP on the FIRM graph gives the feedback shown in Fig. 18, which results in an 88% success probability.

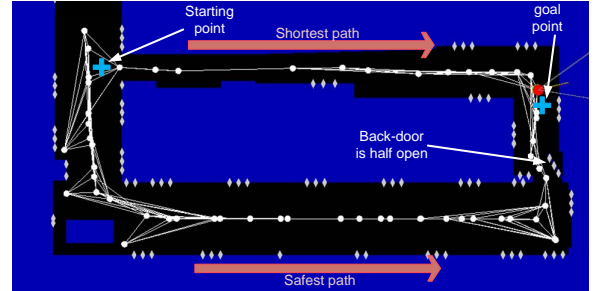


Fig. 16. The environment including obstacles (blue regions), free space (black region), and landmarks (white diamonds). An MAPRM graph approximating the connectivity of free space is also shown (white graph).

C. Online Replanning aspect of SLAP

In this section, we focus on the “simultaneous” part in SLAP, which emphasizes the ability of the robot to replan after every localization update. In other words, in SLAP, the robot dynamically replans based on the new information coming from its sensors.

We look into two important cases to illustrate the effect of online replanning. We first look into a challenging case where the obstacle map changes and possibly eliminates a homotopy

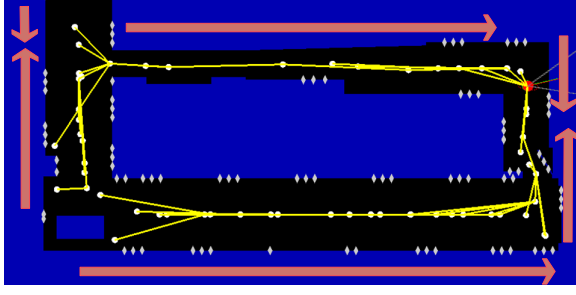


Fig. 17. The feedback tree generated by solving DP on MAPRM is shown in yellow. From each node there is only one outgoing edge, guiding the robot toward the goal defined in Fig. 16. Arrows in pink coarsely represent the direction on which the feedback guides the robot.

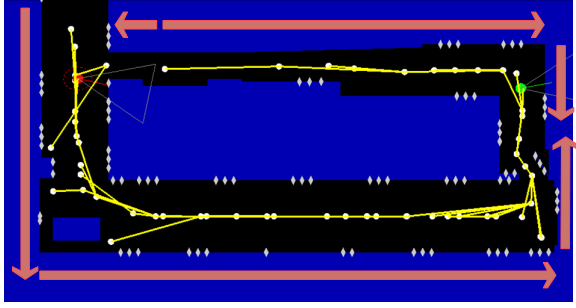


Fig. 18. The feedback tree generated by solving DP on FIRM is shown. As can be seen the computed feedback guides robots through the more informative regions that leads to more accurate localization and less collision probabilities. Arrows in pink coarsely represent the direction on which the feedback guides the robot.

class of solutions. This means a slight deviation in the plan is not sufficient and the planner has to switch to a different homotopy class in real-time, which is not feasible with the state-of-the-art methods in the belief space planning literature. Second, we look into deviations from the path. There, we focus on the kidnapped robot problem as the most severe case of deviation, the solution to which can be applied to smaller deviations as well. Finally, we design a complex scenario that includes changes in the obstacles, small and large deviations, online changes in goal location, etc., and demonstrate the performance of the proposed method on it.

1) Changes in the obstacle map:

Here, we show how enabling simultaneous planning and localization in an online manner, can handle changes in the obstacle map. In this paper, we assume no prior knowledge about the environment dynamics. As a result, we have a simple model for obstacle dynamics: All new obstacles will be added to the map with a large forgetting time of 10 minutes (i.e., almost-permanent forgetting time in our experiments is 10 minutes). The only exception in this model is moving people: if a moving person is detected, a new obstacle will not be added to the map. Instead, we assume there exists a lower-level reactive behavior (e.g., stopping or dodging) in a subsumption-like architecture [15] that suppresses the belief space planner in the vicinity of the moving person. Once the control is back to the SLAP layer, the robot might have deviated from its nominal plan and thus the SLAP layer has to replan to recover from such deviations.

Therefore, the method is very efficient in dealing with persistent/slow changes in the map (e.g., closed-open doors). An important aspect of the method is that it can deal with severe changes that might eliminate or create homotopy classes of solutions. Doors are an important example of this class. If the

robot observes a door is closed (which was expected to be open), it might have to *globally* change the plan to get to the goal from a different passage. This is a very challenging problem for today's belief space planners.

As the first experiment, we consider the environment shown in Fig. 14. The start and goal locations are shown in Fig. 19(a). We construct a PRM in the environment ignoring the changing obstacles (assuming all doors are open and there are no people in the environment). Then we construct a corresponding FIRM and solve dynamic programming on it. As a result, we get the feedback tree shown in Fig. 19(a) that guides the robot toward goal through the back door of room 407. However, the challenge is that the door may be closed when the robot reaches it, and there may be people moving in the environment. Moreover, for different reasons (such as motion blur in the image or blocked landmarks by people or different objects), the robot might mis-detect landmarks temporarily during the run.³ To handle such a change in the obstacle map and replan accordingly, we use the "lazy feedback evaluation" method outlined in Algorithm 4.

Results on physical robots: Figure 19(b) shows a snapshot of our run when the robot detects the change signal, i.e., detects the door is in a different state than its recorded situation in the map. As a result, the robot updates the obstacle map as can be seen in Fig. 19(b) (door is closed). Accordingly, the robot replans; Figure 19(b) shows the feedback tree resulting from the replanning. The new feedback guides the robot through the front door since it detects the back door is closed. The full video of this run provides much more detail and is available in [1].

Comparison with the state-of-the-art: It is important to note that it is the particular structure of the proposed SLAP framework that makes such replanning feasible online. The graph structure of the underlying FIRM allows us to *locally* change the collision probabilities in the environment without affecting the collision probability of the rest of the graph (i.e., properties of different edges on the graph are independent of each other; see Fig. 3). It is important to note that such a property is not present in any other state-of-the-art belief space planner, such as BRM (Belief Roadmap Method) [54], or LQG-MP [68]. In those methods, collision probabilities and costs on *all* edges (the number of possible edges is exponential in the size of the underlying PRM) need to be re-computed. General purpose planner ABT [38] is also not applicable to this setting due to the size of the problem and need to recompute collision probabilities. If an obstacle in the vicinity of robot changes its position, it will change the pdf evolution on a tree branch near the tree root. Accordingly, the whole subtree (including collision probabilities) under that branch needs to be updated.

2) Deviations in robot's location:

In this subsection, we demonstrate how online replanning enables SLAP in the presence of large deviations in the robot's position. As the most severe form of this problem, we consider the *kidnapped robot problem*. In the following we discuss this problem and challenges it introduces.

Kidnapped robot problem: An autonomous robot is said to be in the kidnapped situation if it is carried to an unknown location while it is in operation. The problem of recovering from this situation is referred to as the kidnapped robot problem [24]. This problem is commonly used to test a robot's ability to recover

³Designing perception mechanisms for obstacle detection is not a concern of this research, thus we circumvent the need for this module by sticking small markers with specific IDs on moving objects (doors or people's shoes).

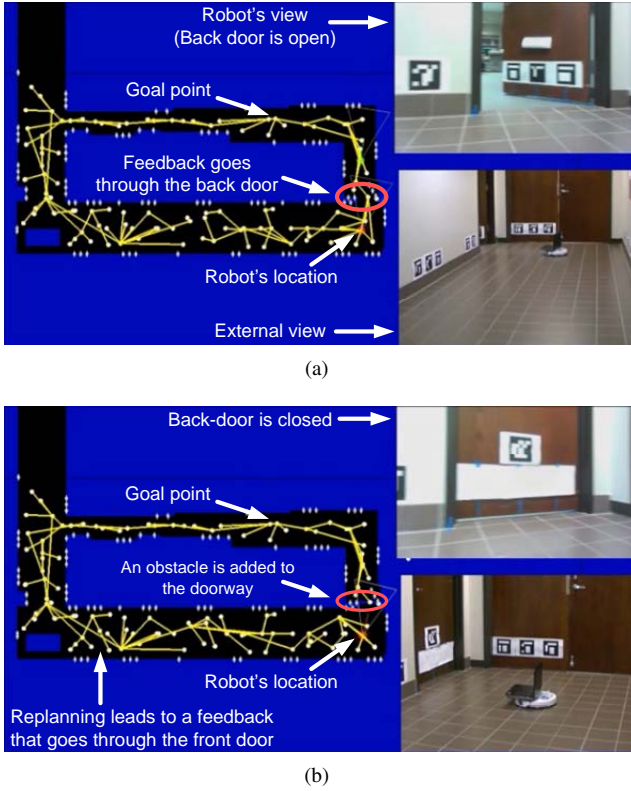


Fig. 19. (a) The back door is open at this snapshot. The feedback guides the robot toward goal through the back door. (b) The back door is closed at this snapshot. Robot detects the door is closed and updates the obstacle map (adds door). Accordingly robot replans and computes the new feedback. The new feedback guides the robot through the front door.

from catastrophic localization failures. This problem introduces different challenges such as (i) how to detect kidnapping, (ii) how to re-localize the robot, and (iii) how to control the robot to accomplish its goal. Our main focus, here, is on the third part, i.e., how to replan in belief space from the new point in the belief space after recovering from being kidnapped. This is in particular challenging because large deviations in the robot's pose can globally change the plan and optimal homotopy class of solutions. Therefore, the planner should be able to change the global plan online.

Detecting the kidnapped situation: To embed the kidnapped situation into the framework in a principled way, we add a boolean observation z^{lost} to the observation space. Let us denote the innovation signal as $\tilde{z}_k = z_k - \bar{z}_k$ (the difference between the actual observations and predicted observation). Recall that in our implementation, the observation at time step k from the j -th landmark is the relative range and bearing of the robot to the j -th landmark, i.e., $^jz_k = (^jr_k, ^j\theta_k)$. The predicted version of this measurement is denoted by $^j\bar{z}_k = (^j\bar{r}_k, ^j\bar{\theta}_k)$. We monitor the following measures of the innovation signal:

$$\tilde{r}_k = \max_j (|^jr_k - ^j\bar{r}_k|), \quad \tilde{\theta}_k = \max_j (d^\theta(^j\theta_k, ^j\bar{\theta}_k)) \quad (17)$$

where $d^\theta(\theta, \theta')$ returns the absolute value of the smallest angle that maps θ onto θ' . Passing these signals through a low-pass filter, we filter out the outliers (temporary failures in the sensory reading). Denoting the filtered signals by \bar{r}_k and $\bar{\theta}_k$, if both conditions $\bar{r}_k < r_{max}$ and $\bar{\theta}_k < \theta_{max}$ are satisfied, then $z^{lost} = 0$, otherwise $z^{lost} = 1$. When $z^{lost} = 0$, we follow the current rollout planner. However, $z^{lost} = 1$ means that the robot is constantly

observing high innovations, and thus it is not in the location in which it believes to be (i.e., it is kidnapped). Once it is detected that the robot is kidnapped, we first replace the estimation covariance with a large covariance (to get an approximately uniform distribution over the state space).

Replanning from kidnapped situation: The rollout-FIRM algorithm can inherently handle such replanning. In other words, the kidnapped situation, i.e., a deviated mean and very large covariance, will just be treated as a new initial belief and a new query, and FIRM rollout will create the best macro-action (funnel) on the fly and execute it. Note that the belief deviation might change the optimal homotopy class and the plan should be updated globally, which makes it challenging for many POMDP planners. Using the proposed rollout planner, the robot just needs to go to a neighboring node from this deviated point. Since the underlying FIRM graph is spread in the belief space, the only required computation is to evaluate the cost of edges that connect the new start point to the neighboring FIRM nodes.

To get safer plans when replanning from $z^{lost} = 1$ situation, we update the rollout planning mechanism slightly: In addition to the new initial belief, we add one more belief node to the graph, as described below. Consider a new kidnapped initial belief $b_0 \equiv (\hat{x}_0^+, P_0)$. Let δ denote the distance between the mean of this new belief \hat{x}_0^+ and the closest mean on the graph nodes. If $z^{lost} = 1$ and δ is not small, the mean belief is far from actual robot position and moving the robot δ meters based on a wrong belief might lead to collision. To ensure that the proposed rollout-based planner can take this case into account, we add a FIRM node b' to the graph at (or very close to) the configuration point $v = \hat{x}_0^+$.

In such a case starting from a deviated belief b_0 with large covariance, rollout planner will take the robot to b' first, which is a belief with the same mean but smaller covariance (i.e., turning in-place or taking very small velocities). The reason is that moving the mean of a distribution with large covariance will lead to high collision probability.

Results on physical robots: Figure 20 shows a snapshot of a run that contains two kidnappings and illustrates the robustness of the planning algorithm to the kidnapping situation. The start and goal positions are shown in Fig. 20. The feedback tree (shown in yellow) guides the robot toward the goal through the front door. However, before reaching the goal point, the robot is kidnapped in the hallway (see Fig. 20) and placed in an unknown location within room 407 (see Fig. 20). In our implementations, we consider $r_{max} = 1$ (meters) and $\theta_{max} = 50$ (degrees). The first jump in Fig. 21 shows this deviation. Once the robot recovers from being kidnapped (i.e., when both innovation signals in Fig. 21 fall below their corresponding thresholds), replanning from the new point is performed. This time, the feedback guides the robot toward the goal point from within room 407. However, again before the robot reaches the goal point, it is kidnapped and placed in an unknown location (see Fig. 20). The second jump in the innovation signals in Fig. 21 corresponds to this kidnapping.

D. Longer and more complex experiments: Robustness to changing goals, obstacles, and to large deviations

In this section, we emphasize the ability of the system to perform long-term SLAP that consist of visiting several goals. The replanning ability allows us to change the plan online as the goal location changes. In this experiment, we consider a

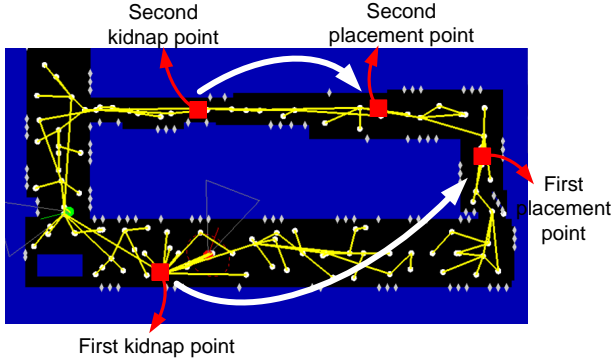


Fig. 20. This figure shows the set up for the experiment containing two kidnappings.

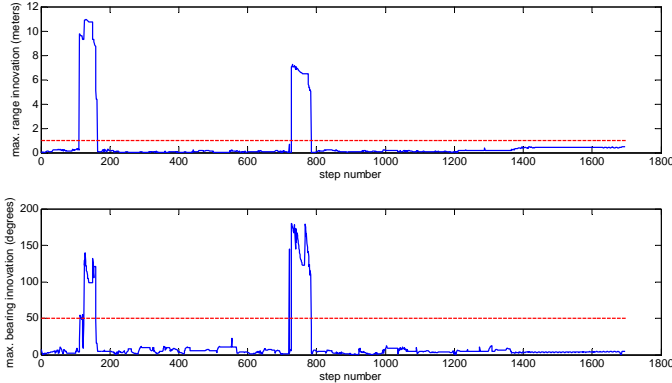


Fig. 21. This figure shows the innovation signals \tilde{r}_k and $\tilde{\theta}_k$ during this run, along with the thresholds r_{max} and θ_{max} (dashed red lines)

scenario in which the user(s) submit a new goal for the robot to reach after it reaches its currently assigned goal. While the robot needs to change the plan each time a new goal is submitted, it frequently encounters changes in the obstacle map (open/closed doors and moving people) as well as intermittent sensor failures and kidnapping situations. Thus, the ability to simultaneously replan online while localizing is necessary to cope with these changes. The video in [1] shows the robot's performance in this long and complex scenario.

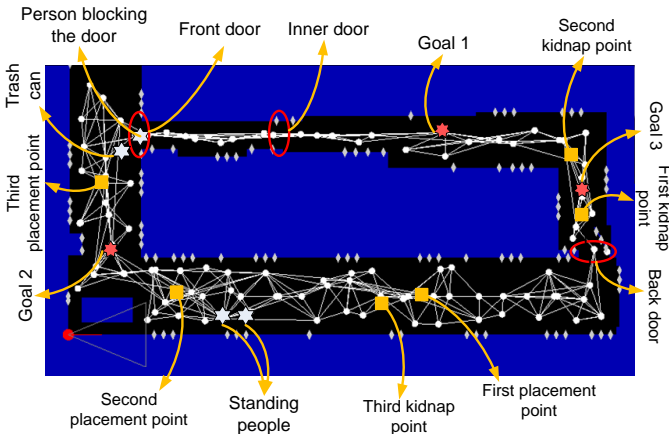


Fig. 22. This figure shows the set up for the longer experiment with a sequence of goals as well as intermediate events and changes in the environment map.

In the following, we provide an itemized description of the

specific steps involved in this run based on Fig. 22. Also, we discuss different changes in the environment with which the robot needs to cope along the way to accomplish its goals. All of the following steps can be seen more clearly in the accompanying video [1].

- 1) The robot begins at the starting point shown in Fig. 22 and aims to reach goal 1 as shown in Fig. 22. Goal 1 is inside room 407. FIRM returns a feedback tree that guides the robot through the back door of 407.
- 2) The robot goes through the narrow passage introduced by the back door (it is half-open). However, before reaching the goal it gets kidnapped (the first kidnap point as shown in Fig. 22). The robot is placed in an unknown location (shown in Fig. 22 by the *first placement point*).
- 3) Observing new landmarks, the robot detects that it has been kidnapped. Accordingly it adds a new node to the graph and replans online. As a result, the feedback guides the robot toward the goal point through the back door again.
- 4) However, in the meantime the back door has been closed. When the robot reaches the vicinity of the back door, it detects that it is closed. Therefore, it updates its map by adding an obstacle at the doorway. Note that the robot will open the door (remove the obstacle) in its map after the forgetting time of 10 minutes. Accordingly, the robot replans a feedback tree that guides it through the front door toward the goal point.
- 5) Along the way, people are moving in the hallway and inside the 407 area. Thus, the robot replans accordingly as it encounters the people. Moving people are reactively avoided and the standing people and static obstacles such as a trash-can (see Fig. 22) temporarily get added to the map as obstacles. Replanning several times to cope with such changes, the robot goes through the front and inner doors and reaches the goal point inside the 407 area.
- 6) After reaching the goal point, another goal (second goal in Fig. 22) is assigned to the robot.
- 7) Replanning for reaching this goal leads to a feedback tree that guides the robot through the inner door, and front door, toward goal 2.
- 8) However, as the robot reaches the vicinity of the inner door, it detects the door has been closed. Therefore, it updates its map and replans accordingly. The replanning leads to a feedback tree that guides the robot toward goal 2 through the back door. Again, along the way robot encounters moving people in the office 407 and in the hallway.
- 9) However, before reaching the goal point, the robot gets kidnapped at the “second kidnap point” as shown in Fig. 22. The robot is placed at a really far-off point (the “second placement point”). Once the robot detects it is kidnapped, it replans and moves slower to gather information. Detecting landmarks, it reduces its uncertainty and continues going toward the goal point.
- 10) After reaching the goal point, the next goal (i.e., third goal) is assigned to the robot (see Fig. 22). Replanning for this goal, leads to a feedback that guides the robot through the front door.
- 11) However, when the robot reaches the front door, it encounters a person standing in the doorway. Accordingly, it replans and decides to go through the back door.
- 12) On the way to the back door, it is again displaced at the “third kidnap point” and placed at the “third placement point”.
- 13) This time, due to the forgetting time, the replanning leads to a path through the front door (the person is not there any more).
- 14) Again, the robot follows the feedback and achieves its goal.

This long and complicated scenario demonstrates how simultaneous planning can lead to a robust behavior in the presence of intermittent model discrepancies, changes in the environment, and large deviations in the robot's location. It is worth noting

that online replanning in belief space is a challenge for the state-of-the-art methods in belief space as they mainly rely on structures that depend on the system's initial belief. Hence, when the system's localization pdf encounters a significant deviation, replanning from the new localization pdf requires the structure to be re-built, which is not a feasible operation online. However, constructing a query-independent graph (the underlying FIRM) allows us to embed it in a replanning scheme such as the proposed rollout policy technique and perform online replanning dynamically to enable SLAP.

VI. METHOD LIMITATIONS AND FUTURE WORK

In this section, we recap the method assumptions and limitations mentioned in previous sections.

Restricted class of POMDPs: As discussed in the previous sections, it is worth noting that the proposed method is not a general-purpose POMDP solver. It provides a solution for a class of POMDP problems (including SLAP) where one can design closed-loop controllers with a funneling behavior in belief space. In the proposed instantiation of the FIRM in this paper, designing funnels requires knowledge about closed-form dynamics and sensor model. Also, the system needs to be locally linearizable at belief nodes, and the noise is assumed to be Gaussian. Further, designing a funnel/controller in belief space requires the uncertainty to be over the part of the state space that is controllable (e.g., the ego-vehicle). For example, the proposed SLAP solution is not applicable to two-player games, where there is no direct control on the opponents motion or sensing.

Combining FIRM with general-purpose online solvers: Most of the general-purpose tree-based POMDP solvers can be combined with FIRM, where the online planners creates and searches the tree and use FIRM as the approximate policy (and cost-to-go) beyond the tree horizon. In particular, when the problem in hand does not satisfy above-mentioned assumptions, one can approximate the problem with a problem that satisfy above assumptions, create a FIRM graph and use it as the base policy. However, in the vicinity of the current belief, one can use general-purpose online POMDP solvers, such as Despot [64], ABT [38], POMCP [60], AEMS [55] that act on the exact problem.

Dealing with dynamic environments: In this paper, we assume no prior knowledge about the environment dynamics. As a result, the simple model we use for new obstacles is: they enter map with a large forgetting time of 10 minutes (with the exception of moving people that do not enter the map and avoided reactively). A more sophisticated and efficient solution can be obtained by learning and modeling changes over time [45] or using some prior on the motion of moving objects. Incorporating such a knowledge in the proposed planning framework is a subject of future work.

VII. CONCLUSIONS

In this paper, we proposed a rollout-policy based algorithm for online replanning in belief space to enable SLAP. The proposed algorithm is able to switch between different homotopy classes of trajectories in real-time. It also bypasses the belief stabilization process of the state-of-the-art FIRM framework. A detailed analysis was presented, which shows that the method can recover the performance and success probability that was

traded-off in the stabilization phase of FIRM. Further, by re-using the costs and transition probabilities computed in the offline construction phase, the method is able to enable SLAP, via online replanning, in the presence of changes in the environment and large deviations in the robot's pose. Via extensive simulations we demonstrated performance gains when using the rollout-based belief planner. As a key focus of the work, we also demonstrated the results of the proposed belief space planner on a physical robot in a real-world indoor scenario. Our experiments show that the proposed method is able to perform SLAP and guide the robot to its target locations while dynamically replanning in real-time and reacting to changes in the obstacles and deviations in the robot's state. Such replanning is an important ability for practical systems where stochasticity in the system's dynamics and measurements can often result in failure. Hence, we believe that the proposed SLAP solution takes an important step towards bringing belief space planners to physical applications and advances long-term safe autonomy for mobile robots.

BIBLIOGRAPHY

- [1] Video of FIRM plan execution on a physical robot in a changing environment. <https://www.youtube.com/watch?v=6cKzcfVDes8>.
- [2] A.-A. Agha-mohammadi, S. Agarwal, A. Mahadevan, S. Chakravorty, D. Tomkins, J. Denny, and N.M. Amato. Robust online belief space planning in changing environments: Application to physical mobile robots. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 149–156, May 2014. doi: 10.1109/ICRA.2014.6906602.
- [3] Aliakbar Agha-mohammadi, Suman Chakravorty, and Nancy Amato. FIRM: Feedback controller-based Information-state RoadMap -a framework for motion planning under uncertainty-. In *International Conference on Intelligent Robots and Systems (IROS)*, SanFrancisco, September 2011.
- [4] Aliakbar Agha-mohammadi, Suman Chakravorty, and Nancy Amato. Nonholonomic motion planning in belief space via dynamic feedback linearization-based FIRM. In *International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, Oct 2012.
- [5] Aliakbar Agha-mohammadi, Suman Chakravorty, and Nancy Amato. On the probabilistic completeness of the sampling-based feedback motion planners in belief space. In *IEEE International Conference on Robotics and Automation (ICRA)*, Minneapolis, September 2012.
- [6] Aliakbar Agha-mohammadi, Suman Chakravorty, and Nancy Amato. FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. *International Journal of Robotics Research*, February 2014.
- [7] Nicolas Kuhnén Arne Sieverling and Oliver Brock. Sensor-based, task-constrained motion generation under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.
- [8] William F Arnold III and Alan J Laub. Generalized eigenproblem algorithms and software for algebraic Riccati equations. *Proceedings of the IEEE*, 72(12):1746–1754, 1984.

- [9] Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A. Ngo. Monte carlo value iteration for continuous-state pomdps. In *WAFR*, volume 68 of *Springer Tracts in Advanced Robotics*, pages 175–191. Springer, 2010.
- [10] Haoyu Bai, Shaojun Cai, Nan Ye, David Hsu, and Wee Sun Lee. Intention-aware online pomdp planning for autonomous driving in a crowd. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 454–460. IEEE, 2015.
- [11] Dimitri Bertsekas. *Dynamic Programming and Optimal Control: 3rd Ed.* Athena Scientific, 2007.
- [12] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Identification and representation of homotopy classes of trajectories for search-based path planning in 3d. In *Proceedings of Robotics: Science and Systems, Los Angeles, CA, June 2011*.
- [13] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 521–528. IEEE, 2000.
- [14] C. Bowen and R. Alterovitz. Closed-loop global motion planning for reactive execution of learned tasks. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014.
- [15] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1): 14–23, Mar 1986.
- [16] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *ICRA*, pages 723–730, 2011.
- [17] Robert R Burridge, Alfred A Rizzi, and Daniel E Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18(6):534–555, 1999.
- [18] L. Carlone and D. Lyons. Uncertainty-constrained robot exploration: A mixed-integer linear programming approach. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1140–1147, May 2014. doi: 10.1109/ICRA.2014.6906997.
- [19] Luca Carlone, Jingjing Du, Miguel Kaouk Ng, Basilio Bona, and Marina Indri. Active slam and exploration with particle filters using kullback-leibler divergence. *J. Intell. Robotics Syst.*, 75(2):291–311, August 2014. ISSN 0921-0296. doi: 10.1007/s10846-013-9981-9. URL <http://dx.doi.org/10.1007/s10846-013-9981-9>.
- [20] H. Carrillo, I. Reid, and J.A. Castellanos. On the comparison of uncertainty criteria for active slam. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2080–2087, May 2012. doi: 10.1109/ICRA.2012.6224890.
- [21] H. Carrillo, Y. Latif, M.L. Rodriguez-Arevalo, J. Neira, and J.A. Castellanos. On the monotonicity of optimality criteria during exploration in active slam. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1476–1483, May 2015. doi: 10.1109/ICRA.2015.7139384.
- [22] S. Chakravorty and R. Scott Erwin. Information space receding horizon control. In *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, April 2011.
- [23] Pratik Chaudhari, Sertac Karaman, David Hsu, and Emilio Frazzoli. Sampling-based algorithms for continuous-time pomdps. In *the American Control Conference (ACC)*, Washington DC, 2013.
- [24] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of robot motion: theory, algorithms, and implementations*. MIT Press, 2005.
- [25] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. Control of wheeled mobile robots: An experimental overview. In *Ramsete*, pages 181–226. Springer, 2001.
- [26] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.
- [27] Tom Erez and William D Smart. A scalable method for solving high-dimensional continuous pomdps using local approximation. In *the International Conference on Uncertainty in Artificial Intelligence*, 2010.
- [28] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practice survey. *Automatica*, 25(3):335–348, 1989.
- [29] Brian Gerkey, Richard T Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.
- [30] Devin Grady, Mark Moll, and Lydia E. Kavraki. Automated model approximation for robotic navigation with POMDPs. In *ICRA*, 2013.
- [31] R. He, E. Brunskill, and N. Roy. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 40:523–570, February 2011.
- [32] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, 25(7):627–643, 2006. doi: 10.1177/0278364906067174.
- [33] V. Indelman, L. Carlone, and F. Dellaert. Planning in the continuous domain: a generalized belief space approach for autonomous navigation in unknown environments. *International Journal of Robotics Research*, 34(7):849–882, 2015.
- [34] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
- [35] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *ijrr*, 2012.
- [36] Ayoung Kim and R.M. Eustice. Perception-driven navigation: Active visual slam for robotic area coverage. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3196–3203, May 2013.
- [37] P. R. Kumar and P. P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice-Hall, Englewood Cliffs, NJ, 1986.
- [38] H. Kurniawati and V. Yadav. An online pomdp solver for uncertainty planning in dynamic environment. In *Proc. Int. Symp. on Robotics Research*, 2013.
- [39] H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems*, 2008.
- [40] H. Kurniawati, T. Bandyopadhyay, and N.M. Patrikalakis.

- Global motion planning under uncertain motion, sensing, and environment map. *Autonomous Robots*, pages 1–18, 2012.
- [41] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3):308–323, 2011.
- [42] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.
- [43] Duan Li, Fucui Qian, and Peilin Fu. Variance minimization approach for a class of dual control problems. *IEEE Trans. Aut. Control*, 47(12):2010–2020, 2002.
- [44] Zakary Littlefield, Dimitri Klimenko, Hanna Kurniawati, and Kostas Bekris. The importance of a suitable distance function in belief-space planning. In *International Symposium on Robotics Research*, 2015.
- [45] Bhaskara Marthi. Robust navigation execution by planning in belief space. In *Proceedings of Robotics: Science and Systems (RSS)*, July 2012.
- [46] R. Munoz-Salinas. <http://sourceforge.net/projects/aruco/>.
- [47] Richard M Murray and Sotale Shankara Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5):700–716, 1993.
- [48] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.
- [49] Giuseppe Oriolo, Alessandro De Luca, and Marilena Vandyttelli. WMR control via dynamic feedback linearization: design, implementation, and experimental validation. *IEEE Transactions on Control Systems Technology*, 10(6): 835–851, 2002.
- [50] Sachin Patil, Gregory Kahn, Michael Laskey, John Schulman, Ken Goldberg, and Pieter Abbeel. Scaling up gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation. In *Algorithmic Foundations of Robotics XI*, pages 515–533. Springer International Publishing, 2015.
- [51] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, pages 1025–1032, Acapulco, Mexico, 2003.
- [52] R. Platt. Convex receding horizon control in non-gaussian belief space. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.
- [53] Robert Platt, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proceedings of Robotics: Science and Systems (RSS)*, June 2010.
- [54] Sam Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *International Journal of Robotics Research*, 28(11-12), October 2009.
- [55] Stéphane Ross, Brahim Chaib-Draa, et al. Aems: An anytime online search algorithm for approximate policy refinement in large pomdps. In *IJCAI*, pages 2592–2598, 2007.
- [56] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32: 663–704, 2008.
- [57] Claude Samson and K Ait-Abderrahim. Feedback control of a nonholonomic wheeled cart in cartesian space. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1136–1141. IEEE, 1991.
- [58] Shridhar K. Shah, Chetan D. Pahlajani, Nicholas A. Lacroix, and Herbert G. Tanner. Stochastic receding horizon control for robots with probabilistic state constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, Minneapolis, September 2012.
- [59] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.
- [60] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, pages 2164–2172, 2010.
- [61] R. D. Smallwood and E. J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.
- [62] T. Smith and R. Simmons. Point-based pomdp algorithms: Improved analysis and implementation. In *Proceedings of Uncertainty in Artificial Intelligence*, 2005.
- [63] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 520–527. AUAI Press, 2004.
- [64] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. DESPOT: Online pomdp planning with regularization. In *Advances in Neural Information Processing Systems*, pages 1772–1780, 2013.
- [65] M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.
- [66] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [67] Noel Du Toit and Joel W. Burdick. Robotic motion planning in dynamic, cluttered, uncertain environments. In *ICRA*, May 2010.
- [68] Jur van den Berg, Pieter Abbeel, and Ken Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *International Journal of Robotics Research*, 30(7):895–913, 2011.
- [69] Jur van den Berg, Sachin Patil, and Ron Alterovitz. Efficient approximate value iteration for continuous gaussian POMDPs. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2012.
- [70] D.H. van Hessem and O. Bosgra. A full solution to the constrained stochastic closed-loop MPC problem via state and innovations feedback and its receding horizon implementation. In *Proceedings of the the Conference on Decision and Control (CDC)*, pages 929–934, Maui, HI, December 2003.
- [71] Steven A Wilmarth, Nancy M Amato, and Peter F Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1024–1031, 1999.